Springer Series in Advanced Manufacturing

Fei Tao
Lin Zhang
Yuanjun Laili

Configurable Intelligent Optimization Algorithm

Design and Practice in Manufacturing

Springer

Springer Series in Advanced Manufacturing

Series editor

Duc Truong Pham, Birmingham, UK

More information about this series at http://www.springer.com/series/7113

Fei Tao · Lin Zhang · Yuanjun Laili

Configurable Intelligent Optimization Algorithm

Design and Practice in Manufacturing

Springer

Fei Tao
Lin Zhang
Yuanjun Laili
School of Automation Science
and Electrical Engineering
Beihang University (BUAA)
Beijing
China

ISSN 1860-5168 ISSN 2196-1735 (electronic)
ISBN 978-3-319-08839-6 ISBN 978-3-319-08840-2 (eBook)
DOI 10.1007/978-3-319-08840-2

Library of Congress Control Number: 2014943502

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Acknowledgments

This book is a summary of Dr. Fei Tao's research in the field of intelligent optimization algorithm and its application from 2009 to 2014 in Beihang University (BUAA). Dr. Tao would like to acknowledge the invaluable cooperation and suggestions, and many collaborators form both China and other countries who have involved his research works on intelligent optimization algorithm.

Especially, thanks for the invaluable contributions from Prof. A. Y. C. Nee, Kan Qiao, and Yue Zhang to the Chap. 6. Thanks for the contributions from Prof. W. T. Liao to the Chap. 7, and Ying Feng's contribution to the Chaps. 7 and 8. Thanks for Prof. Bhaba R. Sarker's contribution to the Chaps. 9 and 10, and Yilong Liu's contribution to the Chaps. 3, 5 and 11.

Some of the contents are published in IEEE Systems Journal, Applied Soft Computing, IEEE Transactions on Industrial Informatics (TII), International Journal of Production Research (IJPR), Computer in Industry, International Journal of Advanced Manufacturing Technology (IJAMT), etc. Thanks all the anonymous reviewers from these journals who have given many valuable and constructive comments to the related researches.

Some contents of this book were financially supported by the following research projects: the Fundamental Research Funds for the Central Universities in China, the Beijing Youth Talent Plan under Grant 29201411, the Nature Science Foundation of China (No.61374199), the National Key Technology Research and Development Program (No.2011BAK16B03), and the Innovation Foundation of BUAA for PhD Graduates (YWF-14-YJSY-011).

Thanks for the help from Prof. Duc Truong Pham, the series editor of Springer Series in Advanced Manufacturing, as well as the reviewers of this book proposal. Thanks for the hard and efficient work by the other peoples in the publisher of Springer.

Of course, our most profound thanks go to our families for their continuous love and encouragements.

Contents

Part I
Introduction and Overview

Intelligent optimization algorithm, which is also called meta-heuristic, is a kind of optimization algorithm that simulates natural phenomena and behaviors with population-based iterations. Its appearance had found a way out for NP-hard problems that are difficult to be solved by many classical deterministic algorithms, and it is able to find feasible suboptimal solutions for complex problems in a relatively short period of time.

The strong versatility, high speed and robustness of intelligent optimization algorithm provide a variety of decision-making solutions for multi-constraint complex numerical and combinatorial optimization problems such as multi-objective service composition, workflow scheduling, manufacturing resource allocation and product quality evaluation and controlling and so on in networked service-oriented manufacturing system. Moreover, it takes advantages of intelligent learning to avoid a large solution space traversal so that the problems can be easily solved. Today, most feasible solutions of these complex manufacturing problems are given by different types of intelligent optimization algorithm. Classic intelligent optimization algorithms, such as genetic algorithm (GA), particle swarm optimization (PSO) and ant colony optimization (ACO) and so on, are also widely reconstructed with various improved and hybrid strategies to adapt different production environments and applications. With highly distributed resources, productions and logistics, more and more improvements or hybridizations are designed to achieve efficient decision-making in every link of product. Intelligent optimization algorithm becomes indispensible in manufacturing.

Therefore, in the Part I of this book, a preliminary introduction of intelligent optimization algorithm and the main optimization problem in manufacturing is presented. This part contains Chaps. 1 and 2. The Chap. 1 presents an overview of the principles, development history and classification of the algorithm. It summarizes the classification of current research emphasis and major trends. The Chap. 2 is a brief overview of complex manufacturing optimization problems, their solution methods and the development of intelligent optimization algorithm in them. From the classification of optimization problems in manufacturing sys-

tem, this chapter lists the major challenges in solving different sorts of the optimization problems. In view of these challenges, typical problem-solving methods are given and the importance of intelligent optimization technology is pointed out. Based on this, we outline the general design patterns and process of intelligent optimization algorithm, and discuss the application challenges, needs and trends of intelligent optimization algorithm in manufacturing systems.

Chapter 1
Brief History and Overview of Intelligent Optimization Algorithms

Up to now, intelligent optimization algorithm has been developed for nearly 40 years. It is one of the main research directions in the field of algorithm and artificial intelligence. No matter for complex continuous problems or discrete NP-hard combinatorial optimizations, people nowadays is more likely to find a feasible solution by using such randomized iterative algorithm within a short period of time instead of traditional deterministic algorithms. In this chapter, the basic principle of algorithms, research classifications, and the development trends of intelligent optimization algorithm are elaborated.

1.1 Introduction

Intelligent optimization algorithm is developed and integrated from a number of relatively independent sub-fields, including the technology of artificial neural networks, genetic algorithms, immune algorithms, simulated annealing, and tabu search and swarm intelligence techniques. As we all know, the current main intelligent optimization algorithms are based on the mode of population based iteration. They operate a population, which represents a group of individuals (or solutions), in each generation to maintain good information in the solution space and find better positions step by step. It is a common method that is independent from specific problems so that it can handle complex optimization problems that are difficult for traditional optimization methods. Their common characteristics are: (1) all operations act on current individuals in each generation; (2) the searching is based on iterative evolution; (3) the optimization can be easily parallelized by multi-population scheme; (4) most of them can give a satisfactory non-inferior solutions close to the optimal solutions, instead of certainly finding the optimal solution; (4) the algorithm is rather random and cannot guarantee the

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_1

efficiency of finding non-inferior solutions. Therefore we simplify the basic process of intelligent optimization algorithms in Fig. 1.1.

Specifically, the first step in problem solving is the encoding design according to the problem environment, character and constraints. Coding is a mapping of problem variables, and it also directly determines the evolution speed of the algorithm. When an appropriate coding scheme is selected, the algorithm will initialize and generate a certain number of individuals to form a population according to the definition of problem variables. Each variable of an individual is randomly generated in the definition domain, and the fitness value of individual in each generation is calculated according to problem's objective function, i.e. fitness function. Then, according to the coding scheme, the variables are mapped to form a chromosome. After initialization and encoding, the algorithm iteration will be started. In iteration, the combination of operators plays a major role, such as selection, crossover, and mutation operator in genetic algorithm, and path finding and pheromone update in ant colony algorithm. Different operators have different effects in the algorithm, so different composition of operators can usually produce different results in solving problem. Through a few operations, some or the entire individuals in the population are changed. By decoding of the new individuals, a new group of solutions can be obtained. Through the evaluation of population according to fitness function (objective function), the best and the worst individual in the population can be picked out. Then we could update the whole population in generation by using random alternative strategy, elitist replacement strategy or merge-based individual optimal selection. After that, the next iteration will be triggered. When the number of iterations reaches a certain ceiling or the satisfactory feasible or optimal solution has already been reached, we can jump out the iteration loop, and output the global best solutions.

In this evolution mode, problem variables and objectives are reflected by coding and fitness functions, while constraints are embodied in fitness function as penalty function or in coding as a bound. The operators are independent to the problems and can be easily implemented. The unified iterative process can make the design, improvement and hybrid research of intelligent algorithms simpler, more intuitive, and more flexible. With iteration-based sub-optimal searching, intelligent optimization algorithm can effectively avoid the combinatorial explosion when solving NP-hard problem. However, not all operators can be arbitrarily combined to form an effective algorithm. Since the evolutionary process is quite random, and operators have different characteristics and limitations in exploration and exploitation respectively, many intelligent optimization algorithms still have some defects such as premature convergence and large result deviation and so on. Therefore, researchers in various fields still keep looking for bran-new coding schemes, operators, good improvements and hybrid forms of intelligent optimization algorithms. All these works are trying to search better feasible solution with limit iterations and limit size of population.

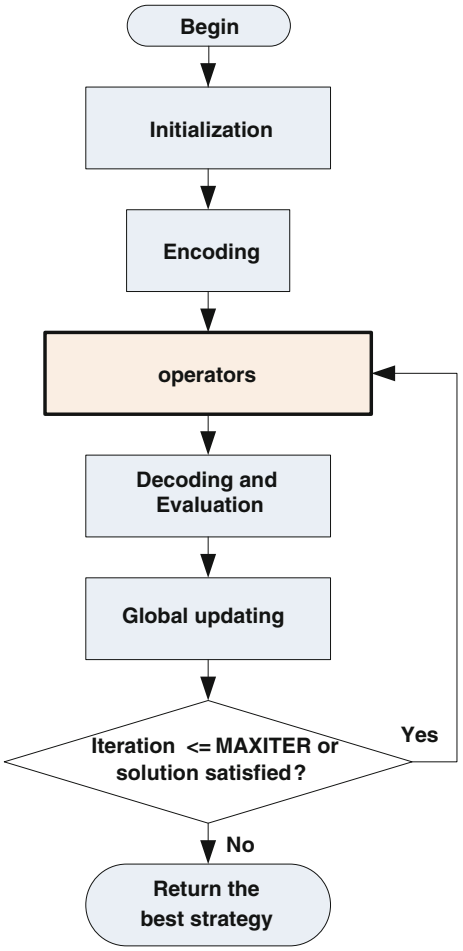


Fig. 1.1 the basic process of intelligent optimization algorithms

1.2 Brief History of Intelligent Optimization Algorithms

With the rapid development of new technology, manufacturing with multi-disciplinary distributed collaboration becomes more and more prevalent in many enterprises. The scale of resources in distribution grows rapidly, and the whole life cycle of manufacturing, including product design, simulation, production, logistics and maintenance, are becoming increasingly complicated. In order to further shorten industry cycle, enhance operation efficiency and improve the utilization of resources and information, many complex problems in macro- and micro-processes should be solved and optimized. From a mathematical standpoint, these problems can also be divided into continuous numerical optimization problems [1, 2] and discrete combinatorial optimization problems [3, 4] according to their variables or the characteristics of the main factors (i.e. the continuity of the solution space), as

well as general optimization problems. For example, the optimization of complex functions, multi linear and non-linear equations in controlling and simulation field, and complex nonlinear programming can be categorized as continuous numerical optimization problems. Typical workflow/task scheduling, service composition optimal selection, collaborative partner selection and resource allocation problems in manufacturing process and system management can be classified as discrete combinatorial optimization problems. Since the quality of these computing and decision-making methods directly determines the efficiency of the whole manufacturing system, experts in all kinds of fields have carried out research to model complex problems under specific conditions and to design of various deterministic algorithm or approximate algorithm for them [5]. In the case of small-scale solution space decisions, various deterministic algorithms can effectively give the optimal solution, and approximation algorithms can also give approximate solution in a shorter time period. However, in reality, most optimization problems are proved to be NP-hard [6–8]. That means no algorithm can find optimal solution in polynomial time. Actually, with the gradually increasing scale of problem, the searching time of traditional deterministic algorithm will grow exponentially, which is quite prone to trigger combinatorial explosion [9]. Likewise, most approximation algorithms have the problem of low approximation ratio, bad versatility and difficulty to meet the requirements of the feasible solutions in the case of solving such large scale problems [10]. Faced with large solution space, both deterministic and approximate algorithms appear to be inadequate.

At this time, the emergence of the intelligent optimization algorithm [11] provides new ideas for this type of large-scale NP-hard optimization problems. Represented by genetic algorithm (GA), simulated annealing algorithm (SAA), ant colony optimization (ACO) and particle swarm optimization (PSO) and so on, this type of algorithm is also known as meta-heuristic. It is formed by simulating biological natural computing skills and the process of evolution. It is also a type of global optimization probabilistic searching algorithm based on the population iterative evolution model. On the other hand, intelligent optimization algorithm can uniformly abstracts problem variables by coding and uses fitness function to represent the objectives. It modifies the population in the solution space by each step iterative operation and evolution with the combination of randomness and guidance to do search. As we have mentioned, it is independent from certain problems and can handle complex optimization problems that are difficult to solve. When solving large-scale NP-hard problem, intelligent optimization algorithm can quickly obtain a set of feasible solutions under the given constrains within a limited time. Instead of finding optimal solution, it uses heuristics to obtain sub-optimal feasible solution and shorten the whole searching process and then improving the efficiency of decision-making. With limited iterations and independent operations, algorithm timeliness would not decrease as the solution space increases and it has good robustness. Therefore, intelligent optimization algorithms have received widespread concern in various fields [12, 13].

Start from the development of early classical optimization algorithms, through careful observation and summary, people have proposed a variety of new

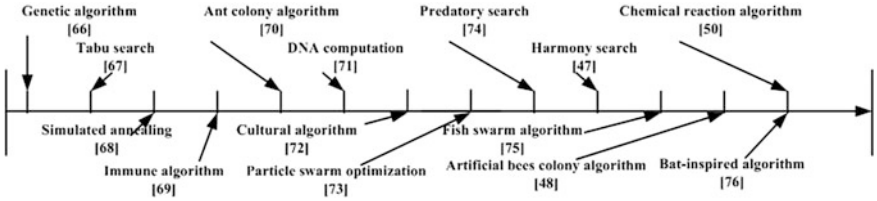


Fig. 1.2 The development of intelligent optimization algorithms [47, 48, 50, 66–76]

intelligent optimization algorithms which are designed to simulate different kinds of nature biological behavior, as is shown in Fig. 1.2. Due to the space limitation, we only list some typical ones, such as Memetic Algorithm (MA) [14], Difference Evolution Algorithm (DEA) and so on [15]. It can be seen that the creation of new intelligent optimization algorithm has not yet stopped and still shows vigorous development trend.

From the results of the application of the very basic intelligent optimization algorithms to many complex numerical problems and standard combinatorial optimization problems (such as the traveling salesman problem and the knapsack problem), a group of researchers have made many improvements from the aspects of algorithm initialization, encoding, operators and evolutionary strategy and produced a series of typical improvement strategy for intelligent optimization algorithm, such as quantum coding improvement [16], adaptive improvement [17], niche strategy improvement [18], prior knowledge-based improvement [19], Pareto search improvement and so on [20]. Considering the problem features, people usually make adjustments on the original basic algorithm in the aspects of exploration, exploitation and the adaptive balance in process respectively and achieved good performance. In addition, after years of study, experts in many fields have found that the hybrid application of many different operators or search mechanisms in different algorithms can greatly enhance the efficiency of problem solving. So hitherto much emphasis has been place on the design and application of hybrid intelligent optimization algorithm. Compared with the design of new operators and improvement schemes, algorithm hybridisation is much simpler, since limited operators stripped from the different algorithms can form a variety of combined intelligent optimization algorithm. After the characteristic analysis of different operators and several tests, a good hybrid intelligent optimization algorithm can be chosen to solve the specific problem.

With a variety of improvements and hybridizations in 40 years' development, the group of intelligent optimization algorithm has been greatly expanded. Therefore, many of the algorithms are applied to engineering practice activities, which have been proved to have good performances in different kinds of benchmark problems. Due to the uniform structure and versatility of intelligent optimization algorithm, users in practice also do some modification and design improvements or hybrid schemes to enhance its performance on specific problem according to particular application environment.

Through the retrieval of intelligent optimization algorithms with several typical keywords during 2001–2012 in "Web of Knowledge" database, we get the number of general literature over the past decade on the intelligent optimization algorithms. According to the following statistical results, a curve with year as abscissa and the annual number of relevant research literatures as ordinate is drawn and shown in Fig. 1.3.

As can be seen from Fig. 1.3, the number of literatures on intelligent optimization algorithm generally has an upward trend in this decade. While in 2009 it reached a peak and had a stable development in the following years, especially in the past 3 years. Actually this research topic is quite popular and abounds great research value, meaning and application prospects because thousands of improvements or hybridizations of intelligent optimization algorithm have been proposed and applied in different fields. However, among the large amount of design and implementation works, whether there exist a large number of duplicate or similar works is still remain unsolved. What's more, with a large number of improved, hybrid and newly proposed algorithm, how to select an appropriate intelligent optimization algorithm to solve the specific problem? These are problems worth studying in depth in the field of intelligent manufacturing.

1.3 Classification of Intelligent Algorithms

As has been explained in the previous section, the number of intelligent optimization algorithms with different evolution mechanisms today has reached to an unparalleled level. Researchers have carried out numerous works on algorithm design for specific problems with different backgrounds and objectives. With different categorizing perspectives, intelligent optimization algorithms can be divided into varying groups. According to the research focuses, the mainstream works on intelligent optimization algorithm can be broadly divided into four categories: (1) algorithm innovation; (2) algorithm improvement; (3) algorithm hybridation; (4) algorithm application. On the basis of literature review in the previous section, we selected 200 of them to conduct research, and draw the approximate percentage of the four categories of institute as Fig. 1.4 shows.

Moreover, according to different algorithm search mechanism, we divide the basic intelligent optimization algorithms into three categories: Evolutionary learning algorithm, neighborhood search algorithm and swarm intelligence algorithm, as is shown in Fig. 1.5.

Evolutionary learning algorithm: Including genetic algorithms, evolutionary programming, artificial immune algorithms, DNA computing and so on, they are formed in accordance with the mechanism of natural learning evolution. Individuals in population are updated by learning from each other in generation according to different heuristics. The most widely ones in this category are genetic algorithm and artificial immune algorithm.

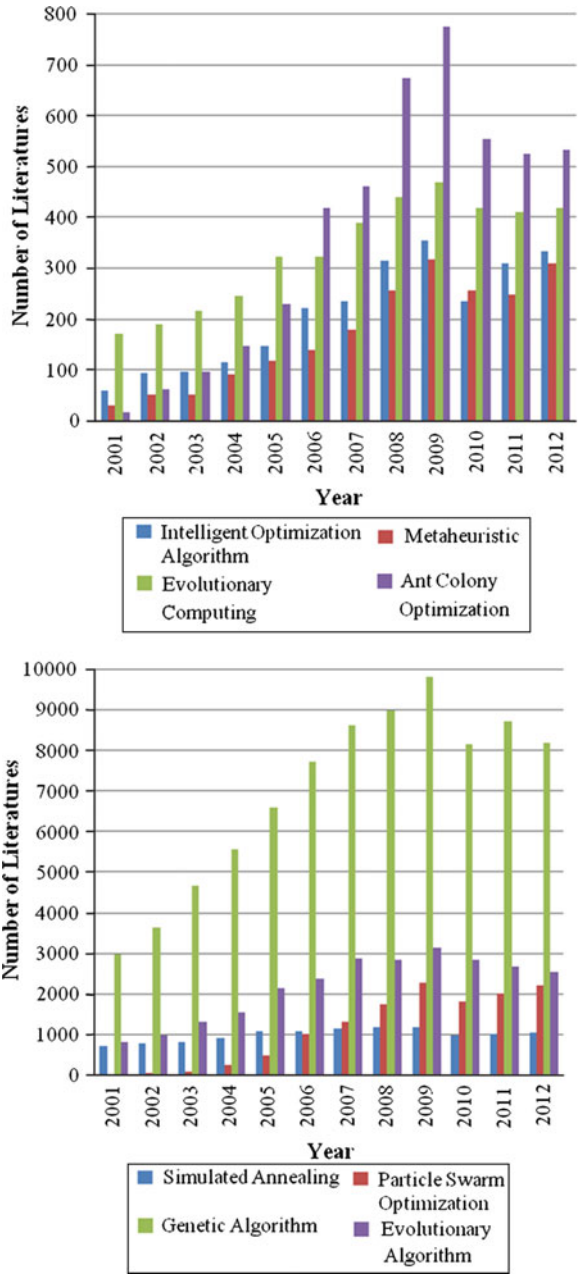


Fig. 1.3 intelligent optimization algorithm research trends in recent years

Neighborhood search algorithm: The most typical ones are simulated annealing algorithm, iterative local search and variable neighbor search. They also belong to local search strategies. Neighborhood search in them are generally implemented by a random or regular changing step and local search step. In searching process,

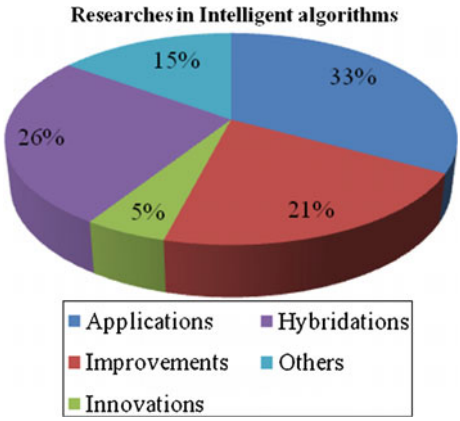


Fig. 1.4 The percentages of the studies on algorithm innovation, improvement, hybridation and application respectively

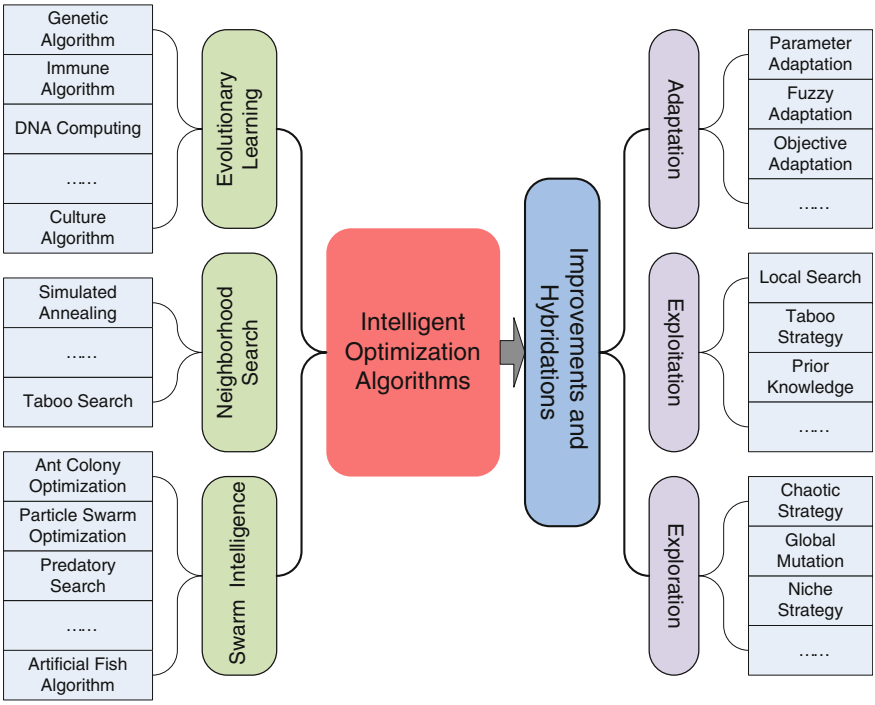


Fig. 5 The classification of intelligent optimization algorithm

additional control parameters or individual acceptance rules in these algorithms are gradually changed to achieve convergence. Thus the main characteristic of this category is the independent local searching of individual. Now most of them are commonly used as an improvement strategy for other algorithms.

Swarm intelligence algorithm: Including ant colony algorithm, particle swarm optimization and artificial fish algorithm and so on, they are designed by simulating self-organizing behavior of social animals (ant colony, bees, flocks of birds, etc.). Normally, they use the broadcasting or transmission of social information extracted by individuals to achieve organizational optimization. At present, the most popular studied algorithms are ant colony algorithm and particle swarm algorithm.

Additionally, with the proposed algorithm shown in Fig. 1.5, a number of improved strategies with different characteristics have been proposed. Researchers conduct their studies from the perspectives of algorithm convergence, exploration, exploitation and stability, aiming to guide the algorithm searching for better near-optimal solutions. From the strategy efficacy [21], we divide these strategies also into three categories: adaptive improvement, exploitation improvement, and exploration improvement.

- Adaptive improvement: Adaptive improvement aims at balancing the search breadth at the prophase and the excavation depth at the late stage. Parameter adaptive improvement, fuzzy adaptive improvement and objective-based adaptive improvement are typical ones.
- Exploitation improvement: Exploitation improvement is primarily to improve the excavation performance of algorithms, mostly manifested by enhancing searching orientation and small-scale traversing.
- Exploration improvement: Global search improvement is designed mainly to enhance the diversity of the population in the algorithmic search, preventing the algorithm from falling into local optimum. Niche strategy, chaos strategy and various mutations are typical ones.

Apart from the above two study branches, there is one more important research focus, i.e. algorithm application. No matter for communication, electronic engineering and mechanical analysis in manufacturing system, or for control and management in manufacturing process, different application object have different requirements on algorithm. In terms of the purpose, the algorithm application object falls into three areas: pedagogical, application and research, as shown in Fig. 1.6.

- Pedagogical: For this purpose, what users want is to quickly understand the basic classification and characteristics of different algorithms. After that, they might move on to their respective principles, mainframes, and performances.
- Application: For application, users care more about selecting appropriate algorithms for addressing the given problem. Theoretical study might be unimportant for them.
- Research: For senior researchers, merely providing a few fixed algorithms is not enough. Beyond that, they need to test and compare more different algorithms under some standard circumstances.

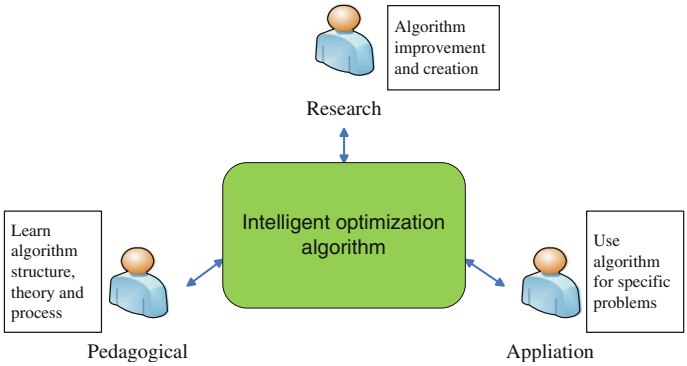


Fig. 1.6 Different design purpose in intelligent optimization algorithm

For different users, the focus is different, and there will be different requirements for algorithm design and application. So how to fully develop the potential and efficiency of the existing intelligent optimization algorithms in different kinds of applications is a problem worth studying.

1.4 Brief Review of Typical Intelligent Optimization Algorithms

Based on different categorizations of intelligent optimization algorithms, this section hence gives brief introduction for several typical algorithms from the perspective of the searching mechanism classification.

1.4.1 Review of Evolutionary Learning Algorithms

The most typical evolutionary learning algorithms include genetic algorithm and immune algorithm and so on. Genetic algorithm simulates biological evolution rules, and uses "survival of the fittest" rules to retain good search information, applies crossover operator to implement information transmission during individuals and employs mutation operator to preserve the diversity of the population. The immune algorithm simulates human body's immune mechanism, using prior knowledge as antigen to guide the whole population search with heuristic directions. Combining with immune selection, individuals with excellent antibody can be obtained and evolution can then be kept. Both of them have shown good performances in representative numerical and combinatorial optimization problems.

1.4.1.1 Genetic Algorithm

Genetic algorithm (GA) is one of the earliest intelligent optimization algorithms, which is relatively mature. The pseudo-code of the most commonly used genetic algorithm can be shown in the following

```
Begin
    t := 0;
    P(t) := InitPopulation();
    Evaluate(P(t));
    While (stop criteria unsatisfied)
        P'(t) = Select(P(t));
        P'(t) = Crossover(P'(t));
        P'(t) = Mutate(P'(t));
        Evaluate(P'(t));
        P(t + 1) = UpdateNewPop(P(t), P'(t));
        t = t + 1;
    End
End
```

where t represents the iteration number, $\mathbf{P(t)}$ represents the population in generation t, and $\mathbf{P'(t)}$ represents the population after one time algorithm operations. Normally, genetic algorithm consists of three basic operators, i.e. selection, crossover and mutation. Since 1975 Holland proposed its complete structure and theory, the improvement of operators for exploration and exploitation and the avoidance of premature convergence are also hot topics in this field.

In terms of encoding, the original method used by Holland is binary coding. But for many applications of genetic algorithms, especially in industrial engineering, this simple coding way is difficult to describe some key properties of problem directly. Thus in recent years, many typical encoding methods are proposed intended to some special or benchmark problems. For instance:

Real number coding [22]: The method is most commonly used coding way in addition to binary coding. It can intuitively express a large amount of continuous and discrete variables, especially suitable to represent variables with large range in genetic algorithm and avoid the process of encoding/decoding. As a result, it can

improve the efficiency and accuracy of genetic algorithm to some extent and reduce the algorithm complexity.

Matrix coding [23]: Matrix coding, in which every variable is represented by multiple gene-bits, is mainly applied to combinatorial optimization problems. It can be used to represent time scheduling sequences, symbol strings, positions in multi-dimensional space and so on clearly and intuitively. In some cases, it can also enhance the searching performance with proper information exchange and diversity keeping. The disadvantage is mainly that it occupies larger memory space and needs encoding/decoding steps which will increase the algorithm complexity.

Quantum coding [24]: This coding method is not only preferred by genetic algorithm, but also widely used in other algorithms. Due to its double-bit coding style, individuals have the characteristics of dimorphism which can largely enhance the diversity of the population in the process of encoding and decoding. It makes individuals randomly cover a wider range of solution space, and obtain improvement of algorithm search performance at the expense of the increase of time complexity.

More than we mentioned, more coding methods are designed and presented as the development of intelligent optimization algorithms, such as dynamic coding, symbol coding, and structure coding and so on.

In the prospect of operators, selection operator plays an evolution guiding role with the rule "survival of the fittest". A variety of improved operators are proposed during research and development, for instance:

- roulette selection strategy
- tournament selection strategy
- sort selection strategy
- random selection strategy.

Crossover operator, as the core operator, is used to recombine gene information, generate offspring, and spread good information. So far, except the single or multi-points crossover, the commonly used methods also include:

- uniform crossover
- shuffle crossover
- crossover with reduced surrogate.

Besides, mutation operator allows a few individuals to jump out of the current states and explore the new area. It plays a role to avoid the premature convergence and enhance the exploration ability of algorithm. In addition to single/multi-point mutation, commonly used methods are:

- uniform mutation
- chaotic mutation
- Gaussian mutation.

By and large, GA has a deep theoretical foundation in both theory and application aspects [25–27]. Other than these improvements, there are also many artificial intelligence strategy-based hybrid genetic algorithms with different forms. Currently, most classic coding methods, typical operators, improvements

and evolution strategies are researched and obtained based on genetic algorithm, and gradually applied to other typical intelligent optimization algorithms. Of course, for different problems, the searching performances of these algorithms are all different and more or less have some drawbacks on searching ability or process time. Hence the researches on genetic algorithm and its improvements for different specific problems are never stopped.

1.4.1.2 Immune Algorithm

The diversity of identification of the immune system has brought a lot of inspiration in the researches of intelligent optimization algorithm. According to immune system and its learning mechanism of diversity recognition, various immune algorithms are designed. The pseudo-code of the typical immune algorithm [28] is as follows:

```
Begin
    t := 0;
    P(t) := InitPopulation();
    Evaluate(P(t));
    While (stop criteria unsatisfied)
        P'(t) = Crossover/Mutate(P(t));
        P'(t) = Vaccination(P'(t) );
        P'(t) = ImmuneSelection(P'(t) );
        Evaluate(P'(t));
        P(t + 1) = UpdateNewPop(P(t), P'(t));
        t = t + 1;
    End
End
```

where t represents the iteration number, **P(t)** represents the population of generation t, **P'(t)** represents the population after one time operations. The immune algorithm contains three basic operators: vaccination, crossover/mutation and immune selection. Priori knowledge of problem is generally extracted as antigen to guide the individual changing. The priori knowledge-based individual changing is vaccination. After that, typical or special designed crossover or mutation operators are employed to make further variation. Finally the immune selection operator tries to

select good individuals with high fitness to realize population updating. The whole process simulates the adaptive production of antibodies in human body's immune system to make the algorithm searching with adaptive convergence and breadth-searching ability. Its main drawback is that the selection of vaccines and vaccination requires deep analysis of the properties and priori information of specific problems, which largely increases the design process and decreases the versatility of algorithm.

In addition, inspired by artificial immune system, there are several other immune algorithms:

- Immune programming (IP) [29].
- Clonal selection algorithm [30].
- Negative selection algorithm [31].

In which immune programming is similar to the immune algorithm, using the diversity and maintaining mechanism of immune system to maintain the diversity of population. In addition, clonal selection algorithm uses the characteristics of adaptive immune response to construct evolutionary selection based on the cloning reaction affinity maturation process, which gets a lot of attention researches. Other than clonal selection algorithm, negative selection algorithm conducts changing and monitoring of the individual alternating probability based on the principle of self and non-self recognition in immune system to realize evolution and optimal selection. The three important principles in negative selection algorithm are: [1] each monitoring algorithm is unique, [2] the monitoring process is probabilistic and [3] the system with robustness should be able to randomly monitor external events rather than search for a known mode. Of course, since the artificial immune systems and its applications still have a long way to go, the design of different sorts of immune algorithms concerning parameter selection and theoretical discussion still has much to be improved.

1.4.2 Review of Neighborhood Search Algorithms

The most typical examples of neighborhood search algorithm are Simulated Annealing (SA) algorithm and Iterative Local Search (ILS) and Variable Neighborhood Search (VNS). We only introduce the first two methods in this chapter. Simulated annealing imitates the annealing process in thermodynamic, achieving optimal selection and convergence process based on random neighborhood search technique. Iterative local search is a combination of local search or hill climbing strategy and general random operation. The evolutionary process of these two methods can be either based on single-individual or on population, and the search strategy can be flexibly altered. Therefore, now they usually play a significant role in the hybridization with other algorithms to improve the overall performance of problem solving.

1.4.2.1 Simulated Annealing Algorithm

In the year of 1982, Kirkpatrick et al. brought the annealing theory into the field of combinatorial optimization, proposing simulated annealing algorithm to solve large-scale combinatorial optimization problems. The essence of this algorithm lies in the simulation of liquid freezing and crystallization process or metal solution cooling and annealing process. If the simulating process is sufficient enough, the algorithm can converge to the global optimal solution with the probability of 1. To maximum problems, the pseudo code is as follows

```
Begin
    t := 0;
    P(t) := InitPopulation();
    Evaluate(P(t));
    While (stop criteria unsatisfied OR T > T_min)
        P' (t) = NeighborSearch(P(t));
        For i = 1 to N
            e := I_i'(t) – I_i(t);
            If (e > 0)
                I_i(t + 1) = I_i'(t);
            Else if (exp(e / T) > random(0, 1))
                I_i(t + 1) = I_i'(t);
            Else
                I_i(t + 1) = I_i(t);
            End
        End
        T = r * T;
        t = t + 1;
    End
End
```

where t is the number of iterations, and N represents the population size. **P(t)** is the tth generation, and **P'(t)** means the new population generated by the neighborhood search operators. $I_i(t)$ and $I_i'(t)$ are two individuals of **P(t)** and **P'(t)**, respectively. T is annealing temperature, $0 < r < 1$ is the cooling rate, T_{min} is the lowest temperature. As can be seen, simulated annealing contains two fundamental operators, neighborhood search and annealing acceptance judgment. It can not only accept optimized solutions, but also accept limited deteriorated solutions with Metropolis principle.

Even though the convergence of the original form of simulated annealing can be proved from a probability perspective, its convergence speed is quite slow. So that many improvements have appeared to improve its performance, including modified cooling schedule [32], heating and annealing with memory [33], two-stage simulated annealing [34], etc. Moreover, due to the randomness and flexibility of the neighborhood search, the annealing acceptance rule is often stripped out to be the improvement of other algorithms, and hence getting a better suitability and search diversity to different problems. Actually, different acceptance functions will bring different influences to intelligent optimization algorithm.

1.4.2.2 Iterative Local Search

Iterative local search [35] is a representative neighborhood search algorithm which was firstly proposed by Lourenco in 2003. It is an extension of local or neighborhood search. In each generation, each individual tries to do local search within a small scope and update itself with better position. Through introducing a perturbation operator, the history information of the previous iteration can be used for guiding and the whole process local traverse scope can be fully enlarged. It is more flexible than the traditional local search. For different problems, the implementation of perturbation and local search operators needed to be redesigned. Its pseudo code is shown below:

```
Begin
    t := 0;
    P(t) := InitPopulation();
    Evaluate(P(t));
    P(t) := LocalSearch(P(t));
    B_est := P(t);
    While (stop criteria unsatisfied)
        P'(t)  = Pertubation(P(t));
        P*'(t) = LocalSearch(P'(t));
        P(t + 1) = AcceptanceCriterion(P*'(t), P'(t));
        t = t + 1;
    End
End
```

where t is the number of iterations, and $P(t)$ represents the individuals in the tth generation. **P'(t)** is the new population generated by perturbation and **P*'(t)** is the new population found by local search. Similar to simulated annealing, Iterative local search also contains acceptance criterion and local search operator. More than that, the perturbation based on current population **P(t)** is introduced to increase the diversity and guide the whole population do neighborhood search in different position.

The standalone use of iterative local search [36] has gotten great success in the fields of combinatorial optimization, such as travelling salesman problem and job-shop scheduling problems. Following that, many hybrid version of iterative local search are presented for scheduling in different environments, such as the hybridization with greedy strategy for unrelated parallel machine scheduling [37] and the hybridization with genetic algorithm for location-routing problem [38] and so on.

Though these neighborhood search methods were proposed relatively early, their ability for solving complex optimization problems is still strong. With the above neighborhood search structures, many deterministic searching algorithms can be partly introduced into the iterative process to enhance the exploitation and find better solutions. It can also be designed as a bridge between traditional deterministic algorithms and the new advanced intelligent algorithms.

1.4.3 Review of Swarm Intelligence Algorithm

Swarm intelligence algorithms search for optimal solution to the problem by individual collaboration and competition in population. The most representative ones are ant colony optimization and particle swarm optimization. Anty colony optimization simulates the pheromone secretion in ants foraging behavior, positioning population with pheromone and taking advantage of pheromone dissipation to avoid population premature convergence. Particle swarm optimization imitates birds' global and individual optimal learning mechanisms, integrating excellent global and individual information and hence implement directed cooperation. To sum up, ant colony optimization is suitable for combinatorial optimization problems such as path finding and scheduling, while particle swarm optimization is more applicable to complex continuous numerical problems.

1.4.3.1 Ant Colony Optimization

In 1990s, Dorigo proposed the first ant colony algorithm—ant system, and meanwhile applied them to the unsolved travelling salesman problem (TSP). From then on, basic ant colony optimization gets continuous development and improvement. Today, these different versions share a same characteristic, i.e., the improved ant detection ability in searching process. The pseudo code of commonly used ant colony optimization is:

Begin

t := 0;

P(t) := InitPopulation();

Evaluate(**P(t)**);

$\mathbf{P_{heromone}(t)}$:= InitPheromone(**P(t)**);

$\mathbf{P_{rior}}$:= InitPriorKnowledge();

While (stop criteria unsatisfied)

$P(t + 1) = FindPath(P(t), P_{heromone}(t), P_{rior})$;

Evaluate(**P(t + 1)**);

$P_{heromone}(t + 1) = UpdatePheromone(P(t + 1), P_{heromone}(t), P_{rior})$;

$t = t + 1$;

End

End

where t is the number of iterations, and $\mathbf{P(t)}$ represents the tth generation. $\mathbf{P_{heromone}(t)}$ and $\mathbf{P_{rior}}$ are the pheromone matrix of the tth generation and priori knowledge information matrix, respectively. They are used to guide ants' path finding behavior. Therefore, it contains two fundamental operators, path finding and pheromone updating, which aim at guiding the population searching by the integration of static priori knowledge and dynamic pheromones which are formed by every individuals' step.

As is shown, ant colony optimization is specifically designed to solve combinatorial optimization problems, and it has concurrency, robustness and positive feedback characteristics. Current mainstream improvements to ant colony optimization include:

- Improvements on foraging behavior (path-finding method)
- Improvements on pheromone updating
- Parameter self-adjustment.

An early representative example of improved method is elitist strategy [39]. Through elite screening, better paths are more likely to be chosen. But if there are too many elites, the algorithm will fall into a local optimum, resulting in the search premature stagnation. In order to overcome these exposed drawbacks, literature [40] proposed an ant colony system, which improved algorithm behavior selection rules and enhanced path pheromone of the global optimum. In the direct improvement on ant colony optimization, MAX-MIN ant system is a typical representative, who modified the way of pheromone updating, and helped to increase the search capabilities of algorithm in the initial stage [41]. Besides, researchers further analyzed and interpreted the invariance and classification of the algorithm in different point of views [42, 43]. Its applications ranged from TSP to quadratic assignment problem (QAP), vehicle routing problem (VRP) and robot path planning. However, although the applications and improvements are relatively diverse, its parameter setting, convergence, effectiveness all come from a large number of experimental results, which not only lack theoretical research like genetic algorithm, but also lack mature methods to guide the process analysis.

1.4.3.2 Particle Swarm Optimization

Kennedy et al. firstly designed particle swarm optimization in 1995. This algorithm conducts searching according to the pursuit of particles to the best individual in solution space, whose process is simple and easy to implement. PSO has simple parameters without complex adjustments, and its implementation is shown in the following pseudo code:

Begin

t := 0;

P(t) := InitPopulation();

Evaluate(**P(t)**);

$\mathbf{p_{best}}$:= the best solution founded by each individual

g_{best} := the best solution founded by the whole population

V(t) := InitVelocity();

While(stop criteria unsatisfied)

V(t + 1) = UpdateVelocity(**V(t)**, $\mathbf{p_{best}}$, g_{best});

P(t + 1) = UpdateLocation(**P(t)**, **V(t + 1)**;

Evaluate(**P(t + 1)**);

Update $\mathbf{p_{best}}$, g_{best} according to **P(t + 1)**;

$t = t + 1$;

End

End

where t denotes the number of iterations, **P(t)** is the tth-generation population, $\mathbf{p_{best}}$ and g_{best} represent the individuals' current best solution matrix and the global best solution, respectively. **V(t)** is called as velocity matrix which stores the updated incremental positions for all dimensions in each generation. Even if particle swarm optimization is divided into two coupled steps, the updating of particles' velocity and the changing of particles' position, it can still be considered as two procedures, which are self learning and global learning according to functions. Updating rate and position according to self optimization and global optimization share the same effects.

From its emergence to the present, particle swarm optimization is rapidly used in function optimization, neural networks, fuzzy systems control and other fields. Most researchers have focused mainly on the aspects of the algorithm structure and performance improvement, which can be classified as:

- Parameter setting
- Algorithm convergence
- Topology architecture
- Particle diversity.

In addition, there are also researches emphasize on population structure and hybridation with other algorithms. In parameter setting, typical examples are the introduction of inertia weight and nonlinear variation of inertia weight proposed [44], etc. From algorithm convergence, M Clerc introduced compression factor to maintain the convergence of algorithms [45]. Besides, J Kennedy's classical research about topology structure significantly affected the system performance [46]. As a new evolutionary algorithm, it is easy to implement and suitable to continuous problems. Yet most of the current researches focus on the applications. The study on algorithm internal mechanism is relatively rare. The immature model parameter settings, the position and velocity structure design rely too much on the experience, whose own structure is overlooked and yet to be perfected.

1.5 The Classification of Current Studies on Intelligent Optimization Algorithm

A variety of intelligent optimization algorithm has appeared one after another, and according to the research focus, existing research can be divided into: algorithm innovation, algorithm improvement, algorithm hybridization, algorithm parallelization and algorithm application. Here's a brief summary of these five aspects.

1.5.1 Algorithm Innovation

The algorithms mentioned above still have a poor diversity and limited searching ability, despite they have brought different improvements to the solution of many complex optimization problems. Inspired by different problems, many researchers are discovering brand new intelligent optimization algorithms to bring new operators to this field.

As is shown in Fig. 1.2, eye-catching algorithms include:

- Memetic Algorithm [14]: from the operators point of view, it is similar to a kind of genetic algorithm with local search, including selection, crossover and mutation, and local search operators;
- Harmony Search [47]: It simulates the timbre reconciliation of musicians, transforming and accepting individual gene-bits with a certain probability. The operating mechanism approximates simulated annealing.
- Artificial bee Colony Algorithm [48]: It selects multiple investigation bees to conduct local search combined with a number of preferred sites, and meanwhile to use the remaining individual to carry on cooperative random search. Bee colony algorithm is a rising star in the field of intelligent optimization algorithm;

- Quantum Evolutionary algorithm [49]: Using quantum coding and quantum-bit operations to achieve a probabilistic search, the coding method of this algorithm is wide used while the operators' application is relatively rare;
- Chemical Reaction Algorithm [50]: Chemical Reactions algorithm takes use of molecular collisions, combinations, split and cross to implement evolutionary change. Although the process of cross and molecular collisions are similar to that in genetic algorithm, this algorithm adds combination and split operators, expressing great potential in some combinatorial optimization problems;
- Plant Growth Algorithm [51]: Similar to Taboo search, it bases on single-individual to select growth point, generating a series of new individuals and selecting the best one to be the substitute of the original one. With a high complexity, it is only proper to continuous optimization problems;
- Group Leadership Algorithm [52]: Similar to bee colony algorithm, it selects leaders according to fitness values, and makes use of cross to achieve in-group local search and inter-group interaction.

What's more, there are also new heuristic evolutionary methods such as Invasive Weed Algorithm [53] and Firefly Algorithm [54], but we would not repeat here. To sum up, it can be found by the algorithms above that most of the operators in new optimization algorithm design are similar or even the same as that in classical genetic algorithm, ant colony algorithm and simulated annealing operators. Actually, only a few of them are brand new operators, and the effectiveness is yet to be further tested by experimental and theoretical proof. The performance of these new algorithms cannot and has never been compared in a standard platform, but only the rule pattern of them has gotten an innovative progress. Therefore, it is well worth of investigating that how to further find out the characteristics and potentials of these new algorithms to achieve higher level efficient search rather than just comparing them with conventional methods.

1.5.2 Algorithm Improvement

Another development branch of new intelligent optimization algorithms is the algorithm improvement. Based on fundamental principles, algorithm initialization, encoding method, operators and evolution strategy have different effects on the process of solution search in most problems. Thus, many researchers draw on the above four aspects to improve existing algorithms.

Algorithm Initialization: The method and application of algorithm initialization can be summarized as random initialization, sequential initialization, initialization with same starting point and priori information-based initialization. The most common used is random initialization, which has relatively less research currently.

Encoding Method: Encoding method had been illuminated in Sect. 1.4.1.1, i.e., the improvement of genetic algorithm. Existing researches concerning new

encoding method are increasingly less, and most mainly focus on typical encoding way according to specific problems.

Operators: The typical ways to improve an algorithm are parameter adjustment and operator adjustment. Since operators usually need different parameters in different problems or in different searching period of one problem, many researches concentrate on the design of population-state based parameter adaptive modification method. In addition, due to the blindness of search, most studies tend to balance the exploration and exploitation of algorithms according to their overall characteristics. Niche strategy, orthogonal strategy and priori knowledge-based strategy are examples in this case. By this mean, many improved operators come into being and can be widely used in different algorithms. By and large, operators are the core of algorithm, thus researches in this field are the most flourished point and have brought many improvements in intelligent optimization.

Evolutionary Strategy: Evolutionary strategy contains Elitist Strategy (i.e. optimal instead of the worst), the Sort Preferred strategy, Competition Strategy, etc. Elite Strategy is the most commonly used one. Although there is not much research regarding evolutionary strategies, the screening of new and old individuals is also an important part, which needs to be further studied.

1.5.3 Algorithm Hybridization

The iterative evolution modes of intelligent optimization algorithms are comparatively unified. Differences only lie in operators, which provide great space for development. Hitherto, according to the overall efficacy of different operators, people can arbitrarily amalgamate the algorithms to make improvements for solving different problems. Consequently, to algorithms that emphasize exploitation, we can introduce operators with more diversity from other algorithms, while to those focus on exploration, we can also add operators with more local mining ability. Moreover, self-adaptive search strategies can be added to improve the performance of hybrid algorithms as well. Given that operators are mainly implemented based on population, when comparing with the improvement of algorithms, the arbitrary combinations of operators in different algorithms are relatively easy and convenient. Therefore, both algorithm beginners and engineering employers can freely select different operators to design various hybrid algorithms and test their efficacy after simply understanding of algorithm iterative modes. Furthermore, how to select suitable algorithms from so many valuable improvements and hybridizations and to practically apply them is a problem that needs insightful discussion.

1.5.4 Algorithm Parallelization

Parallel intelligent optimization algorithms combine the high performance of super computers and the natural parallelism of the algorithms. It can greatly enhance algorithms' processing speed and expansion space. The only parallel strategy of optimization algorithm is population-based parallelism, which means dividing population into sub-group to implement concurrent searching. The earliest studies of parallelization are mainly based on genetic algorithm. From then on, people have proposed many schemes for it. All these works can be classified into three main frameworks, i.e., master-slave parallel model, coarse-grained parallel model and fine-grained parallel model [55]. These three frameworks made possible a variety of parallel intelligent optimization algorithms in recent decades, and the most typical examples are parallel genetic algorithms, parallel particle swarm optimizations and parallel ant colony optimizations.

At present, studies on the parallelism of algorithms mainly focus on three aspects that can largely influence the parallel performance: parallel topology, communicational migration cycle, and the number of communicational migration. Especially in topology structures, there exist single and bi-directional ring topology [56], grid topology [57] and hypercube topology [58] and so on. Presently, even though some research have compared different topology structures and concluded that single bi-directional ring topology was better [59], the role played by different topologies in parallelization is also different. According to the specific circumstances and problems, the performances of different topologies are still to be analyzed with migration cycle and migration number. Overview, there are two issues in parallel studies, which are:

Whether it is possible to design a new parallel mode, expect for population-based parallelism;

Densely connected parallel topology has a large information exchange, strong collaborations, but slow speed, while loosely connected one has smaller information exchange and collaboration, but faster speed.

So the problem of how to embody integrated topology structure to get best search performance urges researchers to conduct deeper analysis and discussion in different application background.

1.5.5 Algorithm Application

Except for the two research branches above, another important point is the application of algorithms. No matter in communication [12], project construction management [13], electronic engineering [60] or manufacturing related fields [61–64], intelligent optimization algorithms have brought varying degrees of efficiency to problem solving. From the standpoint of problem type, algorithm application can be divided into two aspects:

Functions optimization application: The optimization of some typical complex functions is often regarded as the algorithm performance evaluation criteria. Researchers construct many static and dynamic multimodal function such as concave/convex functions and high/low-dimensional functions and consider them as Benchmark to provide comparison for algorithms. And a variety of optimization algorithms aiming at functions optimization are applied to automatic control and power calculation area. In terms of automatic control, intelligent optimization algorithms are commonly used for continuous optimization problems like controller parameter adjustment, simulated control system and control rules learning and control system simulation. It makes possible further out-of-hand regulation and achieves systematic automation and intelligence with higher control accuracy. As to power calculation, intelligent optimization algorithms usually play important roles in the optimization of large-scale multidimensional energy consumption calculation function and circuit loss. It can improve decision-making qualities within a short time, largely lowering electricity power waste.

Combinational optimization application: With the more and more complex multi-domain system, the sizes of a variety of combinatorial optimization problems increase sharply. Thus, many NP-hard problems appear in real situations. The emergences of intelligent optimization algorithms give a way to these problems. For typical combinatorial optimization problems such as the traveling salesman problem, knapsack problem and packing problem,researchers have proposed many different algorithms to try to achieve decision-making efficiency and improvement. Moreover, in real engineering projects, intelligent optimization algorithm is applied to machinery manufacturing, image processing, machine learning and data mining. Among them, the most typical one is resource scheduling in manufacturing management. In fact, the proposal and development of intelligent optimization algorithms can effectively solve the problem of job shop scheduling, production planning and task allocation; In image processing area, intelligent optimization algorithms are often used for pattern recognition and image feature extraction to achieve precise identification and display of the image. While in machine learning, intelligent optimization algorithms often appear in the fuzzy control rule learning, classifier dynamic tuning and the screening and combination of learning rules to improve the intelligence of machine learning. When it comes to data mining, intelligent optimization algorithms can achieve the evolution of search rules and direct search in order to improve the accuracy.

Besides, in their respective fields, researchers have successfully developed various forms of decision-making system based on typical intelligent optimization algorithms such as the genetic algorithm and ant colony algorithm, and have brought much efficiency to field decision-making. From the perspective of optimization algorithm development and application, in studies based on most specific problems, most researchers tend to re-design algorithms or directly use one or several traditional algorithms and improvement strategies according to problem features. In experiments, they usually compare their methods with conventional certain algorithms and the early non-improved algorithms rather than with existing improved studies on intelligent optimization algorithms. As can be seen, there are

fewer relationships between the research focus on algorithm mechanisms and on applications. Therefore, it seems an urgent problem that how to combine the two together to accelerate algorithm development and make it more efficient in engineering area.

1.6 Development Trends

Intelligent optimization algorithms now remain a research focus in the field of artificial intelligence, whose main development direction is towards high efficiency and high intellectualization. However, despite this, when dealing with more and more large-scale complex optimization problems, they are still faced with challenges such as how to take care of multi-users and multi-applications at the same time. Therefore, throughout the development needs of the networked integrated manufacturing system for complex productions, we can summarize the development trend of intelligent optimization algorithms as follows: intelligent, service-oriented, application-oriented and user-centric.

1.6.1 Intellectualization

Intellectualization can be embodied from two aspects. One is that algorithms should have certain functions like parameter self-adjustment and dynamic self-adaptive operations when solving different problems. This kind of quality, which can be defined as function intellectualization, makes possible good performances for different types of problems. The other is that algorithms must have excellent recognition performance during the process of designing, implementing and applying. This quality is defined as application intellectualization.

Currently, many researchers have devoted to the function intellectualization of intelligent optimization algorithms, while the application intellectualization is rarely studied. For application, they tend to simply mix multiple operators, with the help of parameter adjustment and parallelism to adapt varying problems. However, "there is no free lunch" [65]. D.H Wolpert's theory had proven that enforcement of any algorithm to specific area in one aspect always needs additional cost in other aspects, not mention to a multi-suitable algorithm. In summary, great development potential lies in the application intellectualization field, and the problem of how to intelligently integrate different algorithms or operators to cater varying problems is a field that deserves deep investigation.

1.6.2 Service-Orientation

The same as services in networked manufacturing, service-orientation is designed to encapsulate intelligent optimization algorithms in the form of services and provide to clients in different areas. This requires universal interface design for different algorithms and simplified problem interfaces. However, complex issues abound, and their respective attributes are ever changing. So it is difficult to offer universal, service-oriented and efficient algorithms.

Of course, there are very few researches about the servitization of intelligent optimization algorithms, and most researchers only envisaged framework on it, but did not actually begin. Nevertheless, seeing from the rich concepts of networking and cloud and the convenience of service-oriented software and hardware, we could safely draw the conclusion that the service-oriented optimization algorithms are well worthy of studying in future.

1.6.3 Application-Oriented

In the development of intelligent optimization algorithm, application-oriented area can be said is the most important and attractive one in the direction trends. Many researchers have focused on the designing of algorithm based on Benchmark without tests and applications on specific practical problems. Moreover, there are many large-scale problems in reality, and most domain experts are not equipped with optimization algorithm-related knowledge. Among the existing algorithm improvements, only the selecting of suitable intelligent optimization algorithms has made many practitioners overwhelmed. Most applications are more likely to select common algorithms to improve and amalgamate, and only few can devote to design new algorithms to solve certain problems. Therefore, the repetition of many works and the waste of many excellent research studies on intelligent optimization algorithms are unavoidable.

Due to this situation, many researchers are considering to build a uniform platform so that many high-efficient algorithms and their improvements can be categorized and collected to provide reference to users in different industry. But how to build such a flexible, convenient and application-oriented platform is exactly both a research focus and a research difficulty.

1.6.4 User-Centric

We have mentioned in the analysis of algorithm classification that intelligent optimization algorithms have three kinds of users: algorithm beginners, algorithm

researchers and algorithm users. From the initial goal of algorithm design, algorithms should be user-centric, and different users' demands vary.

To algorithm beginners, they hope to understand the classification and features of many basic algorithms, and to learn their principles, structures and processes.

To algorithm researchers, they not only need to study existing algorithms and their improvements, but also be able to conduct tests and theoretic analysis in a standard platform according to different problems.

To algorithm users, they may have no deep understanding to algorithms, and in real applications, they just need to find out corresponding optimization algorithms and one or two improvements to encode and design. The study and improvement of algorithms are too complicated to them.

Intelligent optimization algorithms' research are rarely concerned about user-centric problem. But to the expansion of intelligent optimization algorithms, this problem can never be underestimated. Therefore, user-centric algorithm design and improvement is another potential mainstream trend in future.

1.7 Summary

Intelligent optimization algorithms are beneficial to the following problems.

Large-scale optimization problems, which are difficult to get solution with deterministic algorithm on limited computing resources.

Real time decision problems, which require the algorithm with high time efficiency. Suboptimal solutions are generally acceptable in such problems. When dealing with such problems, intelligent optimization algorithms could bring very high solving efficiency.

In this chapter, referring their functions and features, we divided intelligent optimization algorithms into evolutionary study, neighborhood Search and population intelligence and also respectively elaborated their research outlines and basic principles. This classification is preliminarily achieved based on main functions, and although operation differences lie between different kinds of algorithms, their mechanisms are probably the same. Moreover, many improvements are actually the hybridization of different algorithms, so there is no significant boundary between them. The research of algorithms has been abundant, but the pace of development has not stopped. We briefly summarized the development trend of algorithms in this chapter. In prospect, the theories, features and applications of intelligent optimization algorithms are still the research focus in many fields of human endeavor, which belong to a key technology in interdisciplinary decision optimization.

References

1. Nocedal J, Wright SJ (2006) Numerical optimization. Springer, Berlin
2. Bonnans JF, Gilbert JC, Lemarechal C, Sagastizabal CA (2006) Numerical optimization: theoretical and practical aspects. Springer, Berlin
3. Papadimitriou CH, Steiglitz K (1998) Combinatorial optimization: algorithms and complexity. Dover Publications, Mineola
4. Schrijver A (2003) Combinatorial optimization. Springer, Berlin
5. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. J ACM Comput Surv (CSUR) 35(3)68–308
6. Garey MR, Johnson DS (1990) Computers and intractability: a guide to the theory of NP-completeness. W. H Freeman and Co, San Francisco
7. Ullman JD (1975) NP-complete scheduling problems. J Comput Syst Sci 10(3):384–393
8. Gawiejnowics S (2008) Time-dependent scheduling. Springer, Berlin
9. Karp RM (1986) Combinatorics, complexity, and randomness. Commun ACM 29(2):98–109
10. Kann V (1992) On the approximability of NP-complete optimization problems. Royal Institute of Technology, Sweden
11. Talbi EG (2009) Metaheuristics: from design to implementation. Wiley, New york
12. Ribeiro CC, Martins SL, Rosseti I (2007) Metaheuristics for optimization problems in computer communications. Comput Commun 30(4):656–669
13. Liao TW, Egbelu PJ, Sarker BR, Leu SS (2011) Metaheuristics for project and construction management—a state-of-the-art review. Autom Constr 20(5):491–505
14. Moscato P (1989) On evolution, Search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech Concurrent Computation Program
15. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):341–359
16. Tao F, Zhang L, Zhang ZH, Nee AYC (2010) A quantum multi-agent evolutionary algorithm for selection of partners in a virtual enterprise. CIRP Ann Manufact Technol 59(1):485–488
17. Shi Y, Eberhart RC (2001) Fuzzy adaptive particle swarm optimization. In: Proceedings of the 2001 congress on evolutionary computation, vol 1, pp 101–106
18. Horn J, Nafpliotis N, Goldberg DE (1994) A niched pareto genetic algorithm for multiobjective optimization. In: Proceedings of the 1st IEEE congress on evolutionary computation, vol 1, pp 82–87
19. Wang DW, Yung KL, Lp WH (2001) A heuristic genetic algorithm for subcontractor selection in a global manufacturing environment. IEEE Trans Syst Man Cybern Part C 31(2):189–198
20. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
21. March JG (1991) Exploration and exploitation in organizational learning. Organ Sci v2(1):71–87
22. Tsoulos IG (2008) Modifications of real code genetic algorithm for global optimization. Appl Math Comput 203(2):598–607
23. Zhang G, Gao L, Shi Y (2011) An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Syst Appl 38(4):3563–3573
24. Zhang G (2011) Quantum-inspired evolutionary algorithms: a survey and empirical study. J Heuristics 17(3):303–351
25. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Kluwer Academic Publishers, Boston
26. Whitley D (1994) A genetic algorithm tutorial. Stat Comput 4(2):65–85
27. Schmitt LM (2001) Theory of genetic algorithms. Theoret Comput Sci 259(1–2):1–61
28. Wang L, Pan J, Jiao LC (2000) The immune algorithm. ACTA Electronica Sinica 28(7):74–78
29. Wang L, Pan J, Jiao LC (2000) The immune programming. Chin J Comput 23(8):806–812

30. de Castro LN, Von Zuben FJ (2002) Learning and optimization using the clonal selection principle. IEEE Trans Evol Comput 6(3):239–251
31. Hofmeyr SA, Forrest S (2000) Architecture for an artificial immune system. Evol Comput 8(4):443–473
32. Noutani Y, Andresen B (1998) A comparison of simulated annealing cooling strategies. J Phys A: Math Gen 41(31):8373–8385
33. Ali MM, Torn A, Viitanen S (2002) A direct search variant of the simulated annealing algorithm for optimization involving continuous variables. Comput Oper Res 29(1):87–102
34. Varanelli JM (1996) On the acceleration of simulated annealing. University of Virginia, USA
35. Lourenco HR, Martin O, Stutzle T (2003) Iterated local search. Int Ser Oper Res Manag Sci 57:321–353 (Handbook of Metaheuristics. Kluwer Academic Publishers)
36. Lourenco HR, Martin O, Stutzle T (2010) Iterated local search: framework and applications. Int Ser Oper Res Manag Sci 146:363–397 (Handbook of Metaheuristics, 2nd edn. Kluwer Academic Publishers)
37. Fanjul-Peyro L, Ruiz R (2010) Iterated greedy local search methods for unrelated parallel machine scheduling. Eur J Oper Res 207(1):55–69
38. Derbel H, Jarboui B, Hanafi S, Chabchoub H (2012) Genetic algorithm with iterated local search for solving a location-routing problem. Expert Syst Appl 39(3):2865–2871
39. Dorigo M, Maniezzo V, Colorn A (1996) The ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern 26(1):29–42
40. Dorigo M, Gambardella M (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66
41. Stutzle T, Hoos HH (2000) MAX-MIN ant system. Future Gener Comput Syst 16(8):889–914
42. Birattari M, Pellegrini P, Dorigo M (2007) On the invariance of ant colony optimization. IEEE Trans Evol Comput 11(6):732–742
43. Martens D, De Backer M, Haesen R, Vanthienen J, Snoeck M, Baesens B (2007) Classification with ant colony optimization. IEEE Trans Evol Comput 11(5):651–665
44. Chatterjee A, Siarry P (2006) Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. Comput Oper Res 33(3):859–871
45. Clerc M, Kennedy J (2002) The particle swarm-explosion, stability and convergence in a multidimensional complex space. IEEE Trans Evol Comput 6(2):58–73
46. Kennedy J, Mendes R (2002) Population structure and particle swarm performance. In: IEEE 2th proceedings of evolutionary computation, pp 1671–1676
47. Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimization algorithm: harmony search. Simulation
48. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39(3):459–471
49. Platel MD, Schliebs S, Kasabov N (2009) Quantum-inspired evolutionary algorithm: a multimodel EDA. IEEE Trans Evol Comput 13(6):1218–1232
50. Lam AYS, Li VOK (2010) Chemical-reaction-inspired metaheuristic for optimization. IEEE Trans Evol Comput 14(3):381–399
51. Wang C, Cheng HZ (2008) Optimization of network configuration in large distribution systems using plant growth simulation algorithm. IEEE Trans Power Syst 23(1):119–126
52. Daskin A, Kais S (2011) Group leaders optimization algorithm. Mol Pheys 109(5):761–772
53. Mehrabian AR, Lucas C (2006) A novel numerical optimization algorithm inspired from weed colonization. Ecol Inform 1(4):355–366
54. Yang XS(2008) Nature-inspired metaheuristic algorithms. Luniver Press
55. Muhlenbein H, Schomisch M, Born J (1991) The parallel genetic algorithm as function optimizer[J]. Parallel Comput 17(6–7):619–632
56. Yang HT, Yang PC, Huang CL (1997) A parallel genetic algorithm approach to solving the unit commitment problem: implementation on the transputer networks. IEEE Trans Power Syst 12(2):661–668
57. Fukuyama Y, Chiang HD (1996) A parallel genetic algorithm for generation expansion planning. IEEE Trans Power Syst 11(2):955–961

58. Xu DJ, Daley ML (1995) Design of optimal digital-filter using a parallel genetic algorithm. IEEE Trans Circ Syst 42(10):673–675
59. Matsumura T, Nakamura M, Okech J, Onaga K (1998) A parallel and distributed genetic algorithm on loosely-coupled multiprocessor system. IEICE Trans Fundam Elect Commun Comput Sci 81(4):540–546
60. Yeung SH, Chan WS, Ng KT, Man KF (2012) Computational optimization algorithms for antennas and RF/microwave circuit designs: an overview. IEEE Trans Industr Inf 8(2):216–227
61. Tao F, Zhao DM, Hu YF, Zhou ZD (2008) Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system. IEEE Trans Industr Inf 4(4):315–327
62. Tang KS, Yin RJ, Kwong S, Ng KT, Man KF (2011) A theoretical development and analysis of jumping gene genetic algorithm. IEEE Trans Industr Inf 7(3):408–418
63. Lo CH, Fung EHK, Wong YK (2009) Intelligent automatic fault detection for actuator failures in aircraft. IEEE Trans Industr Inf 5(1):50–55
64. Hur SH, Katebi R, Taylor A (2011) Modeling and control of a plastic film manufacturing web process. IEEE Trans Industr Inf 7(2):171–178
65. Wolpert DH (1997) W G Macready (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
66. Holland J (1975) Adaptation in natural and artificial systems. The University of Michigan Press
67. Glover F (1989) Tabu search. ORSA J Comput 1(3):190–206
68. Kirkpatrick S, Gelatt CD, Vechi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
69. Farmer JD, Packard NH, Perelson AS (1986) The immune system, adaptation, and machine learning. Physica D 22(1–3):187–204
70. Dorigo M (1992) Optimization, learning and natural algorithms. Ph.D. Thesis, Politecnico di Milanno
71. Adleman LM (1994) Molecular computation of solutions to combinatorial problem. Science 266(5187):1021–1024
72. Reynolds RG (1994) An introduction to cultural algorithms. In: The 3rd annual conference on evolution programming
73. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: IEEE international conference on neural networks
74. Linhares A (1998) State-space search strategies gleaned from animal behavior: a traveling salesman experiment. Biol Cybern 87(3):167–173
75. Li XL (2003) A new intelligent optimization algorithm—artificial fish school algorithm. Ph.D. Thesis, Zhejiang University, China
76. Yang XS (2010) A new metaheuristic bat-inspired algorithm. Nature inspired cooperative strategies for optimization (NICSO 2010), Springer, Berlin, p 65–74

Chapter 2
Recent Advances of Intelligent Optimization Algorithm in Manufacturing

Due to its good versatility and independence, intelligent optimization algorithm has largely shortened the time of decision-making in large-scale optimization problems of manufacture. However, lower searching time often conflicts with the searching accuracy in most cases. To improve the problem solving capability, research in intelligent optimization algorithm based on different domain characteristics never stopped. From the view of manufacturing, this chapter classified and comprehensively analyzed all kinds of manufacturing optimization problems and their general methods, illustrated the application features and challenges of intelligent optimization algorithm in manufacturing, and summarized the development needs and trends of intelligent optimization algorithm in the field of manufacturing system.

2.1 Introduction

First of all, the application process of intelligent optimization algorithm in manufacturing engineering consists of five main parts, as shown in Fig. 2.1, problem modeling, variable encoding, operator design, simulation and algorithm implementation. Differs from pure algorithm design, the most critical part of algorithm application is problem modeling and variable encoding. Then the design of operators in algorithm depends largely on the specific environment and coding ways.

Problem modeling: The core of modeling is using variables and formulas to concisely and comprehensively express the three main elements of problem—variables, objectives and constraints—according to the environment and requirement. Moreover, the priori knowledge, environmental parameters and the relationship between variables should be given in concise mathematical expression.

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_2

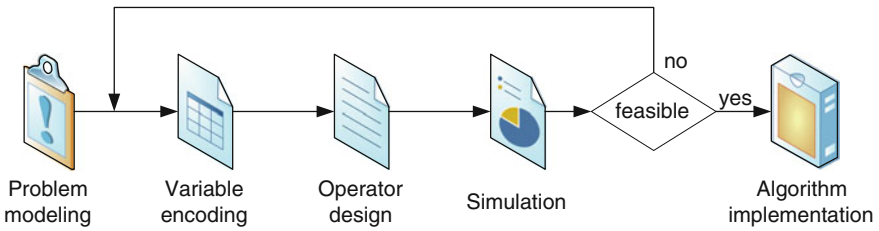


Fig. 2.1 The application process of intelligent optimization algorithm

Variable encoding: Encoding scheme is the link between problem and intelligent optimization algorithm. It is the basis for operators in algorithm to search in the solution space of problem. Different encoding schemes have different levels of randomness and then make the algorithms searching with different capability.

Operator design: With population-based iteration, operators, such as crossover, mutation and so on, need to be selected and designed according to the above encoding scheme. It decides the evolutionary direction of population and the whole searching way of algorithm. Different kinds of operators have different ability of exploration and exploitation and suitable for different sorts of problems. Thus in this step, we should especially focus on the balance of the two ability in the algorithm.

Simulation: Because of the randomness of intelligent optimization algorithm, simulation is the most effective way to verify the algorithm performance with theoretical analysis. Moreover, parameters need to be tuned based on several experiments. If the expected performance is reached, the algorithm can be adopted and applied; if not, we should return and reanalyze the encoding scheme or the operators for adjusting the specific problem.

Algorithm implementation: After the design and simulation, the algorithm can then be developed in practical systems for application.

Based on such a unified process, intelligent optimization algorithm is applied almost in everywhere. It can be seen that there are mainly three types of application objects in manufacturing field: management of manufacturing process, control/simulation for manufacturing system and product/element design and analysis, as shown in Fig. 2.2.

- Management of manufacturing process: It covers the continuous process modeling and discrete workflow management of design, machining and transportation in production line. It is central line for the whole life cycle of manufacturing. Thus it can be called as process optimization.
- Control/Simulation for manufacturing system: It includes the design of manufacturing control system, manufacturing simulation and supervision of production line. Only high efficient control and simulation will guarantee the efficient operation of the whole manufacturing system. It is a kind of system optimization.

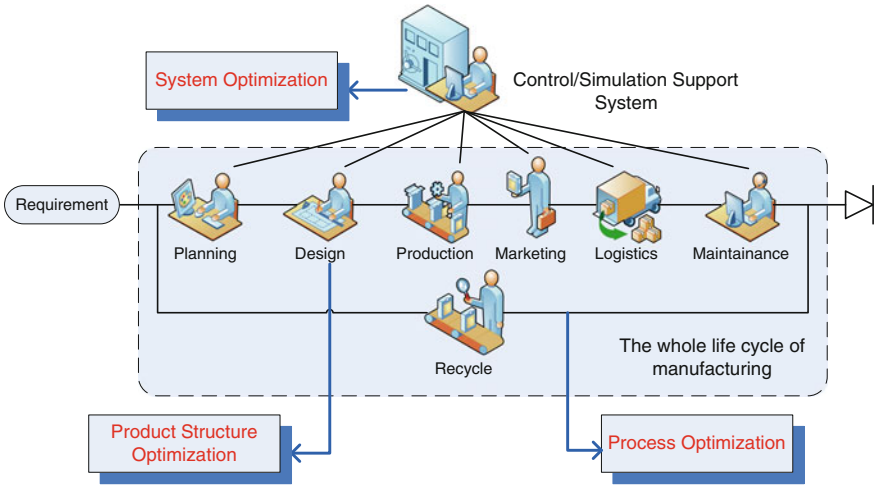


Fig. 2.2 Three main optimization objects in manufacturing

- Product/element design and analysis: It contains the structure design and modeling and finite element analysis of product. It is the core object of manufacturing. Optimization in this category is known as structure optimization as well.

Problems in these three objects have their own particular characteristics and are all along with high complexity. Though a plenty of intelligent optimization algorithms are designed for them, there are still various degrees of difficulties and challenges in the optimization for these three types of objects. On determining how to choose the most beneficial algorithms for different application objects in accordance with problem characteristics, and how to apply and incorporate those algorithms with general optimization methods in practical systems, further discussion is presented next.

2.2 Classification of Optimization Problems in Manufacturing

Problems in manufacturing include single-objective and multi-objective ones. In the light of the attributes of decision variable, problems can be simply divided into continuous numerical optimization and discrete combinatorial optimization as well. From the perspectives of both optimization targets and decision variables, we primarily divide the problems into five categories: numerical function optimization, parameter optimization, detection and classification, combinatorial scheduling, and multi-disciplinary optimization.

2.2.1 Numerical Function Optimization

Numerical function optimization refers to searching for optimal solutions of nonlinear multivariate complex equations. Typical problems in this category include function optimizations in modeling of manufacturing process and complex finite element analysis for product structure design. The variables are usually continuous and the objectives are multi-modal numerical functions.

Generally, numerical function optimization problems often appear in manufacturing process optimization and product structure optimization. The solutions are primarily the suitable values of global characteristics such as cutting or milling speed, and feed rate, process loads and structure length of product. Complex environment always bring about multiple relative parameters and constraints in the problem model. Thus it has the characteristics of large solution space, dispersive and narrow feasible solution regions, complex objective functions and high dimension of variables. Take the structure design of manufacturing part as an example, the decision variables are generally structure sizes, and the objectives are maximizing the key loads and minimizing resource consumption. With nonlinear relationship among stress and strain of the part, material consumptions and the structure sizes, differentiation and integration are both involved in the objective functions, which make the multimodal functions difficult to solve.

In continuous space, when most of the multivariate functions are linear, mathematical programming is commonly used in solving equations. When functions are complex but have small solution space, software such as ANSYS and CAD are often used to simulate. We may find the peak value by means of mathematical modeling and programming. However, with large-scale solution space and nonlinear multivariate functions, when most classical method requires much longer solution time, intelligent optimization algorithm can come in handy. Additionally, because of the uncertainties in the model parameters selection and the machining index tuning, intelligent optimization algorithm with invariance and independence can better adapt to solve these problems. Thus, in recent years, genetic algorithm and particle swarm algorithm, which are suitable for continuous numerical optimization, are applied in many kinds of structure optimization and manufacturing process optimization. And these intelligent optimization algorithms are usually combined with classical deterministic algorithms to optimize the model in two steps or optimize the model under the guidance of classical deterministic algorithms to improve the solutions. Reference related to numerical function optimization in manufacturing can be found in [1–9].

2.2.2 Parameter Optimization

Parameter optimization generally refers to the selection of optimal empirical parameters in complex manufacturing system or process control optimization. In manufacturing system, most parameters such as material and machining properties

have big influence on the manufacturing process and systems. With large uncertainty and complexity, it is hard to build theoretical model to calculating the optimum value of these parameters for different situations. Thus they are mostly extracted and solved independently with nondeterministic algorithms.

Parameter optimization often exists in manufacturing process optimization and system optimization. Differs from numerical function optimization, parameter optimization is the optimization of one specific part or local key point, though the global environmental factors of system or process are considered. The parameters involved are highly dynamic and context-relative. For example, in the process of manufacturing such as casting or milling, the variables to be solved are machining force indexes, control time interval, load variance (upper bound and lower bound), etc. And the aim is the lowest loss, the highest manufacturing speed and the highest machining quality. In the process of optimization, objective functions are usually not able to be set out, and the real-time demand of system control is high, which makes the problem more difficult.

To solve this, parameter optimization has two solutions. If the objective functions can be formulated, parameter optimization is often solved by classical function optimization algorithm, or intelligent optimization algorithm when the solution space is huge. Otherwise, we can only simulate the system or related processes, and take the output as the target value. The parameters can be solved by simple feedback when the solution space is small, or by intelligent optimization algorithm when the solution space is huge or highly dynamic. Most of the recent studies of parameter optimization are developed in the two aspects above, and the main research and development solution is dividing and reducing the feasible empirical region of parameters, and then taking the method of integrating the simulation and tools to find suitable values of them. As a new and convenient decision-maker, the intelligent optimization algorithm is widely used in studies. For recent studies, refer to [10–19].

2.2.3 Detection and Classification

Detection generally refers to determine whether the condition and variation of an entity or event are beyond normal by features. Classification refers to define the category of the entity or event by features as well. Both of them contain the process of feature-extracting, pre-training and state-judgment. Therefore, they can be classified into one group and all follow the training process according to quantities of samples.

Detection is usually reflected in the fault diagnosis of manufacturing control/simulation system and supervision of manufacturing process and so on. Classification is often embodied in signal analysis of electrical system and model state of machining part, etc. Those problems have the characteristics of scattered samples distribution and uncertain features influence. They are widely exist in manufacturing process, system and structure optimization, but have less research than

parameter optimization in manufacturing field. Take the fault diagnosis of machining process as an example, optimization variables are generally the influence weight of several relative features, and the aim is to identify whether a fault exist in the specific case and which kind of fault it is as accurately as possible. Specifically, it is a determining process in which the influence weights of several relative features are trained with samples, and the status of process or objects are detected according to these weights. In a similar way, the classifications are trying to identify the states of objects with feature weights trained from large-scale samples. Problems in this category are slightly similar to parameter optimization mentioned above. In many cases the target function can not be obtained, and we can only make decision according to the output of system or process simulation.

No matter for training or recognition, detection and classification problems are generally solved by some approximate iterative algorithms when the target function can be obtained, or by intelligent optimization algorithm when the solution space is huge, which is quite similar to the parameter optimization as well. If the target function is difficult to obtain, we may simulate the system or related processes in iterations, and take the output as the evaluation criteria. In recent years, most of the studies in detection and classification focus on solving the problems by support vector machine, decision tree and neural network and so on, among which neural network is the most typical one. While there is increasing number of studies in neural network, the application of them in manufacturing field is quite limited. Currently, because of the empirical limitations and complexity of classifier such as neural network, the research is developing mainly in the integration of classifier and other optimization algorithms and their collaborative application in detection and classification. For related studies, refer to [20–30].

2.2.4 Combinatorial Scheduling

Combinatorial scheduling is the most typical combinatorial optimization problem in manufacturing system. It is a reasonable distribution and management of missions, resources and processes. Combinatorial scheduling here includes process planning, job shop scheduling, task scheduling and resource allocation, which schedules the manufacturing process, assembly line, manufacturing services and machines respectively. It is a kind of discrete management and optimization in manufacturing.

Therefore, the variables of combinatorial scheduling are generally integer. The targets are minimizing the task execution time and energy consumption in global or local workflow, and maximizing the quality (such as maintenance and reliability) and the efficiency of production or calculation. The constraints usually are the limits of resource capability, task size and other QoS indexes. The model is simple, while the solution space is typically huge. In addition, the variances and restrictions of variables are complex, which make the feasible solution space more narrow, so as to make the optimization harder. For example, in job shop

scheduling problem, most of the studies aim at shortest completion time of part machining. The steps of machining, the number of machine tools in each step, and their machining capability are known, and the distribution strategy of each machine tool in each step of machining is to be solved. The process is complicated, but the target function is simple. When comprehensively considering multi-QoS and multi-objectives, optimal solutions are always hard to get.

For such problems, when the solution space is not huge, integer programming and dynamic programming are often used in solving. When the solution space gets bigger, the use of deterministic algorithms will always lead to the combinatorial explosion. In most cases, sub-optimal solutions are acceptable in combinatorial scheduling, and short decision time is required. Hence, most researchers pay more efforts on the application of intelligent optimization algorithm in combinatorial scheduling problems. And in manufacturing field, intelligent optimization algorithm has become the most applied method in combinatorial scheduling. Due to its typicality, some of the combinatorial scheduling problems such as job shop scheduling and task scheduling have been used as benchmarks of combinatorial optimization for different researchers to test and analyze the optimization algorithms they designed. The improvements and application of intelligent optimization algorithm in combinatorial scheduling by existing researchers are mostly concentrating on two kinds: algorithm hybridation and encoding scheme design. For related studies, refer to [31–40].

2.2.5 Multi-disciplinary Optimization

Multi-disciplinary optimization refers to the combining modeling and analysis of problems with multi-objectives and constraints in different disciplines such as control/mechanical collaborative design, and realizing multi-disciplinary collaborative decision making. At present, networked and collaborative manufacturing system has being greatly developed. Therefore, the whole life cycle of manufacturing can be connected in network, so as to realize control and mechanical collaborative design, machine and monitoring synchronize execution. Multi-disciplinary optimization then becomes more and more important for collaborative work. With the widely research in integrated manufacturing and service-oriented manufacturing, it gradually develop into one of the typical types of problems in industry.

The variables of multi-disciplinary optimization problems mostly include both discrete and continuous ones. Currently the studies related to this kind of problems are very few, and most of them are based on multi-disciplinary collaborative simulation and solved by multi-step decisions. With more and more complex manufacturing system, it can be embodied in all aspects of process, system and structure optimizations. Because many constraints and objectives come from different fields and the relationships among these factors are complex, transformation and simplification are indispensable. On the other hand, simplification is

obtained based on the loss of modeling accuracy. The complexity of such problems is obviously huge. For example, in the control/mechanical collaborative optimization, not only the stability and the efficiency of the control system need to be guaranteed, but also the applicability and the portability of the mechanical model need to be improved. Therefore, the variables to be solved usually include the parameters of control system and the critical sizes of part mechanical models. The objective functions are multiple efficiency indicators of the collaborative works, such as material cost, energy consumption and control efficiency, etc. It can be seen that the multi-disciplinary optimization problems are the combination of the above four kinds.

Multi-disciplinary optimization is the least studied one in the problems above. Solution methods for it are mostly based on empirical adjustment and experimental simulation. Although most of the existing studies focus on the objects in manufacturing process, there are also some multi-disciplinary problems in the design and simulation of product structure, and system management and control. Now the methods of onside decision or multi-step optimization in collaborative manufacturing are inevitably not thorough enough. Therefore multi-disciplinary optimization becomes a big challenge and developing trend with the development of advanced manufacturing system. For more information, refer to [41–44] for references in recent years.

2.2.6 *Summary of the Five Types of Optimization Problems in Manufacturing*

With large literature review, the above five types of optimization problems can be mapped into the three typical objects in manufacturing as shown in Fig. 2.3. Most common scenarios are contained in the classifications. Among them, manufacturing process optimization covers all five kinds of problems, manufacturing system optimization includes three kinds of problems (parameter optimization, detection and classification, combinatorial scheduling), and product structure optimization contains three kinds, numerical function optimization, detection and classification and multi-disciplinary optimization, as well.

According to random selection of the most related 100 literatures in the last 3 years, it can be found out that combinatorial scheduling is studied the most in manufacturing optimization. It accounts for nearly half of optimization research. There are designs and applications for various kinds of certain and uncertain algorithms targeted to combinatorial scheduling, which covers every steps of manufacturing process and the management of manufacturing system. Then, in the next place, the numerical function optimization and the parameter optimization have a close number of studies. In numerical function optimization, the finite element analysis and structure optimization is the majority, and mainly target to the analysis of various kinds of product designs. In parameter optimization,

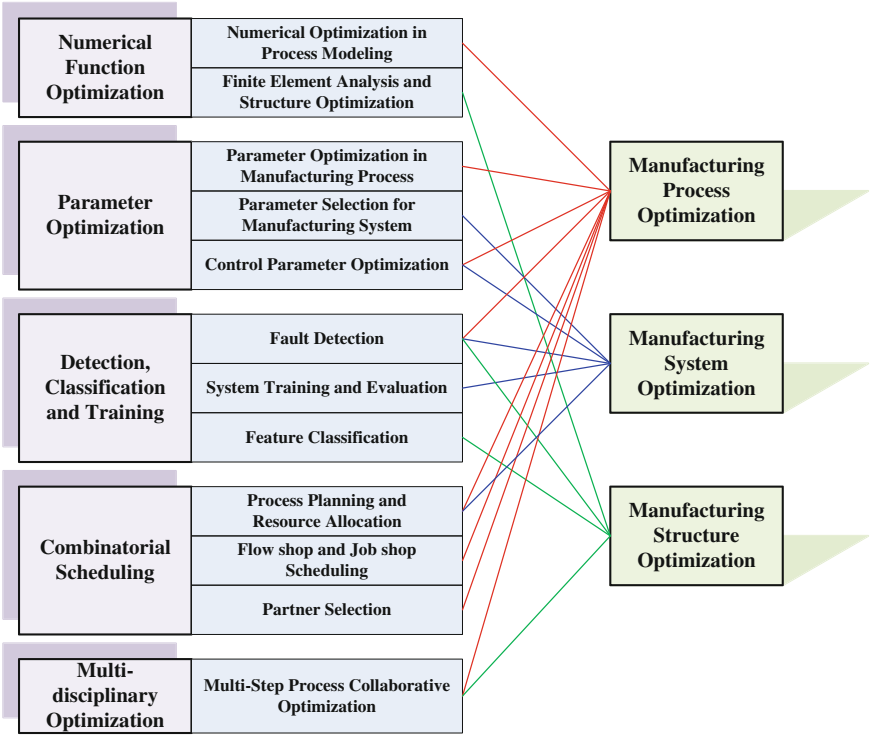


Fig. 2.3 The mapping relationship between typical problems and manufacturing objects

because of the similar adjustment schemes, the studies on parameter tuning in process and parameter alternation in system control are evenly distributed. The three kinds of problems account for nearly 90 % of the studies of manufacturing optimization. On the other hand, there are not many studies on detection and classification in manufacturing field. In this kind of problems, the studies are more on diagnosis and detection, and less on system training and evaluation. The application of feature-based classification in structural optimization is lesser. It is clear that the classification is not highly concerned in manufacturing field yet. Finally, although multi-disciplinary optimization appears more and more in the design of manufacturing process and structure, its complexity makes it the least studies in all these kinds of optimization problems. It is not highly concerned, and the most of the existing studies separate these problems in to several steps and optimize them individually. However, in the above five kinds of optimization problems, the multi-disciplinary optimization problem is one of the most urgent problems. Because of the one-side independent decisions with multi-steps, the decisions are usually inaccurate and the efficiency is not high. A lot of research is required in the modeling of associated features in different disciplines and comprehensive optimization among several disciplines in manufacturing process,

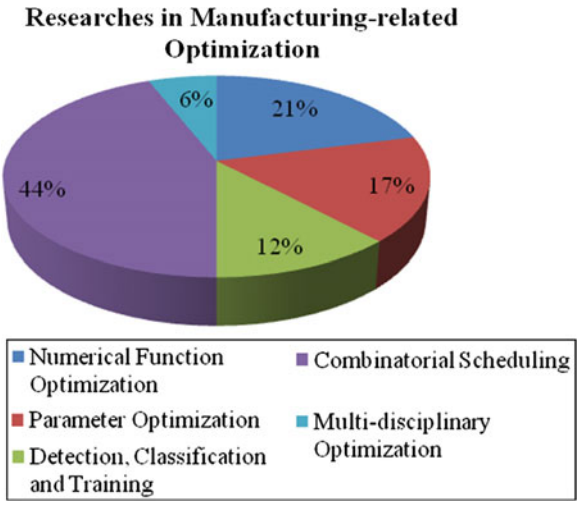


Fig. 2.4 The sample research results on the five types of problems in manufacturing

structure and product design. The sample research results on the problems above are shown in Fig. 2.4.

2.3 Challenges for Addressing Optimization Problems in Manufacturing

After the above analysis, we can see that optimization problems in manufacturing are countless and various. Researchers conducted plenty of works on solving them with different point of view. Nevertheless, those problems with the characteristics of large-scale, multimodal functions and NP-hard are still hard to solve. With widely applied nondeterministic methods, researchers and engineers are taking many attempts to improve the solution quality and reduce optimization time. In this section, we mainly classify the challenges faced in the problems into different categories and analyze them with some existing solution schemes.

The challenges of optimization problems in manufacturing can be divided into seven kinds: Balance of multi-objectives, Handling of multi-constraints, Extraction of priori knowledge, Modeling of uncertainty and dynamics, Transformation of qualitative and quantitative features, Simplification of large-scale solution space, Jumping out of local convergence.

2.3.1 Balance of Multi-objectives

We care the objectives the most in optimization, including the main objectives and the secondary objectives. Most of the problems have more than one objective. It is

unavoidable that several objectives are conflicted with each other. In the case that all objectives are unreachable at the same time, balance of multi-objectives generally refers to the average consideration of multi-objectives according to their weights during optimization process.

To normal multi-objective problems, some typical solutions are listed as follows:

(1) Transformation to single-objective problem: It means to combine the objectives into a single function according to their weights, and solve the problem with only one target function. The weights of different objectives in the unique function are set in experience with specific environment. It is a traditional way with low efficiency.
(2) Transformation to constraints: This method considers the main objectives only and tries to transform the secondary objectives to constraints. It mainly takes some main objectives as optimization goals, and takes the minimum requirements of the other objectives as constraints to solve the problems.
(3) Pareto optimization: It considers all objectives at the same time by Pareto non-dominate sorting scheme. The solution is non-dominate only if all of its objective values are better than others. With a Pareto convex set used to collect those non-dominate solutions, both the main and the secondary objectives are evenly considered. Though the optimization of the main objectives is often restricted by the secondary ones, it is the mostly applied method in recent years.

Currently, engineers mostly use the first two methods according to the actual situation, and the researchers are mostly exploring and studying the third method. Among those methods, the first one is the simplest, and it is the earliest method for multi-objective problems. Because the weight of each method is decided by actual condition, the model is usually empirical and has a narrow application scope. It ignores many optimal solutions after weighting. The second method is more flexible than the first one, but it also requires the experience and environmental factors to decide the strength of constraints, which is the weight of secondary objectives, either. It is more adaptive to general decisions with different algorithms, but the transformation of the objectives to constraints brings us a multi-constraint problem, which is more complex. The third method uses the concept of equilibrium in the game theory, improves the other two methods by avoiding the influence of experience. It can give a series of equilibrium optimal solutions taking advantages of non-dominate sorting and the Pareto convex set. However, because of the complexity of its algorithm design, and the uncertainties brought by selecting the solutions in the Pareto convex set according to actual environment, it is rarely applied in engineering, and the studies and comparisons of the algorithms based on this method are not very clear.

It is thus clear that balance of multi-objectives is one of the big challenges in optimization. Now more and more engineers are trying to apply the Pareto thought to practice. How to implement low complexity determination of Pareto optimal solutions and how to select better solutions in the Pareto convex set are both key

bottlenecks in manufacturing. For balance of multi-objectives in manufacturing problems, refer to references [45–56].

2.3.2 Handling of Multi-constraints

Besides the variables and objectives in decision-making, the constraints produced by the relationships between variables and parameters are one of the direct reasons to complicate problem. In problem modeling, most of the constraints exist in practice are abandoned for simplification. When applying to real production system or process, that may cause low accuracy or even mistakes in decision making. Thus how to suitably handle multi-constraints in accordance with system environment is one of the most important problems in manufacturing optimization.

Following the previous section, the objectives in the problem can be transformed to constraints. When the number of constraints increases, the constraints can be transformed into the objective functions as well. For handling of multi-constraints, there are several specific ways:

(1) Constraints as penalty function: It means to transform the constraints as a penalty function and multiply or add it with the objective functions. If the constraints are not exceeded, the value of the penalty function is 1 or 0, or it becomes a huge value to make the value of objective functions unacceptable, so as to make the solution abandoned.
(2) Bounds checking: In this method, constraints are independently stored as searching rules. Each solution generated in searching process is checked whether there are out of the restrictions or not, if yes, it will be discarded and replaced by a new one.
(3) Branch-and-bound: It is a classical method which narrows the domains of variables, and divides the solution space into several branches and then reduces the searching range. It is also a preprocessing procedure for searching. The only drawback of it is its high complexity and high dependence in the preprocessing for specific problem.

In addition, there are many other strategies like transforming the constraints into heuristic information or objective functions. Traditional engineering mostly uses the first two methods listed above to deal with the problems. After extensive development of algorithmic search, branch and bound, at present, is gradually used in engineering, which brought many benefits. In general, the first method is simple in design, and has got strong versatility. It can quickly filter out the solutions which do not meet the conditions in the space. But it could easily lead to a loss of feasible solutions and inevitable useless search in large solution space. The second method avoids the influence of deciding by experience, but item-by-item checking in optimization process will lead to greater algorithm time complexity. The third one, which narrows the solution space by branch and bound, is one of the mostly popular methods at present. It can greatly decrease the searching complexity, but

brings preprocessing consumption on the other hand. It needs in-depth analysis of actual problems, and with complex problems, it is hard to define the best bounds of solutions with constraints, which leads to a complex design process and low versatility.

Now with the development of intelligent manufacturing system, the handling of multi-constraints in manufacturing problems tends chiefly to design versatile and automatic processing scheme and simplification way for multi-constraints to minimize the searching complexity. For the handling of multi-constraints in manufacturing problems, refer to recent literature [10, 34, 57, 58, 59, 60, 61, 62, 63, 64].

2.3.3 Extraction of Priori Knowledge

For solving a variety of complex problems in manufacturing system and process, research and application also tend to extract the priori knowledge of problem aims at instruct the algorithm to faster searching. Typical examples are the use of prioritization according to priori status of tasks which enables the algorithm find suitable solutions faster, and the selection of nearest neighbor according to priori information in path optimization. It can be down as a kind of greedy strategy. The extraction of priori knowledge has become an important way of solving the problems. When facing various changing problems, the extraction of priori knowledge needs to be conducted in line with the actual environment and features of problems. The versatility of extraction method is low, and improperly designed method will directly cause wrong search direction and then get poor or even wrong solutions.

Currently, on the one hand, the extraction of priori knowledge usually applied in artificial immune systems, artificial neural network systems and intelligent systems based on Agent. By designing the priori knowledge of specific problems, it may perform the rule-based reasoning and prediction to achieve a fast or efficient optimization. On the other hand, it may coordinate with approximation algorithms or intelligent optimization algorithms for complex problems solving, which enhances the searching direction of the algorithms. The extraction of priori knowledge is usually achieved by obtaining the local interactions between variables and objectives. The common factors considered for the extraction can be classified as follows:

(1) Influence of single variable to single objective: Considering only one variable with one critical objective, the interaction between them is calculated as priori knowledge for searching.
(2) Influence of single variable to multi objectives: Considering one key variable with part of objectives, the relationship between the variable and multi objectives are weighed and connect together as priori knowledge.

(3) Influence of multi variables to single objective: Considering multi variable with one key objective, the correlation between multi variables and the objective are weighed and merge together as priori knowledge.
(4) Influence of multi variables to multi objectives: Comprehensively considering part or all of the variables and objectives simultaneously, the priori knowledge is the relationships of the variables and objectives or a reasoning rules for them.

For the calculating of different kinds of influence relations and change of status, we may design an evaluation function as the measurement of priori knowledge, or blur the relations and status and design the mapping between fuzzy priori knowledge and variables. In addition, we can predict the priori knowledge by intelligent training and reasoning according to the existing features and data of simulation systems or models. Now because the lack of research and theoretical analysis in the extraction of priori knowledge especially in nondeterministic optimization, the applications of priori knowledge in manufacturing engineering are much less. An important and difficult point is the way of simplifying and universalizing the priori knowledge extraction and applying them in the widely in actual systems. For the optimization based on priori knowledge, refer to recent literature [65–75].

2.3.4 Modeling of Uncertainty and Dynamics

Problems in manufacturing are all highly uncertain and dynamic. The uncertainty mainly refers to the randomness of characteristics and constraints in the problems, which means that only the range of them can be determined as most time, but the specific values in a period can't be determined. The dynamics refers to the property that the characteristics and constraints of problems are changing with time. The values can be determined only in a period, but they will change gradually. In most manufacturing systems, researchers and engineers always simplify the uncertainties and dynamics of problem to certain values, which will make the design and application of algorithms more convenient. But the simplification will bring inaccuracy and instability. To improve the stability and solving efficiency, the uncertainties and dynamics are accepted as key considerations.

In general, there are several methods to deal with the uncertainty and dynamic nature of the problems in manufacturing:

(1) Replicated simulation: This method is mainly for the modeling of uncertainty. It takes repeated measurements to obtain the mean value and variance of uncertain parameters. Then conduct a number of decisions in a small range around the value to get a set of good solutions. It can be applied in all algorithms but is quite time consumption. Because few tests can not cover all situations, solutions obtained are often inaccurate.

(2) Description with fitting function: This method can be either for the solving of uncertainty or dynamics. From mathematical point of view, it obtains the fitting functions of uncertainty or dynamic by capturing the relation between the actual environment and the variation rules of uncertain or dynamic parameters.
(3) Cyclical forecasting: It is primarily for the modeling of dynamics. It refers to predicting the variation characteristics of the problems at regular intervals. Predicting rules is also conducted according to some test or fuzzy relation among problem features and the environment.
(4) Feedback control: This method can be applied to deal with both uncertainty and dynamics. It does not need to analyze the characteristics of problem and its environment in advance. It refers to design an adaptive feedback control strategy in optimization algorithm to automatically adjust the decision making parameters with variant problem characteristics during the optimization process. It can be seen that this scheme is generally carried out with multi-period problem simulation.

Engineers commonly use the first method according to actual situation, while researchers mostly focus on the design and application with the last three methods, in which the second method and the fourth one are the most typical. In the four methods, the first one is a kind of brute-force methods. It is the earliest processing method of uncertainties and dynamics without mathematical analysis. The second one requires a theoretical basis and practical understanding of the actual problem, and it is more flexible than the first method. However, the design of the second method is harder. The third method conducts regular testing and estimation to the problem by typical predictor and corrector, which solve the design difficulties of the second method. But the prediction time during optimization directly increase the time complexity of algorithm in most cases. The fourth method borrows the idea in control theory and uses the problem states supervised in each period as feedback to design an adaptive strategy which can control the algorithm parameters so as to adapt different situation and obtain good solutions. The design of feedback regulation is simpler than the fitting function, and it is more versatile, but it has the same problems in the design of adaptive control rules as the second method.

Now the accurate modeling of uncertainty and dynamics is still a difficult problem, which is a direct reason of the inefficiency in the decision-making of manufacturing system engineering. Among those methods, cyclical forecasting is quite appropriate for the modeling of dynamics according to the change of time, and has great potential. Feedback control is more fit to the modeling of uncertainty. On the whole, the development of algorithm with the consideration of uncertainty and dynamics which can adaptively adjust the variances in problem is a major trend. For the modeling of uncertainty and dynamics, refer to the references [76–87].

2.3.5 Transformation of Qualitative and Quantitative Features

No matter in manufacturing system simulation or process modeling, qualitative analysis needs to be done at the beginning. Then how to transform the qualitative parameters and variables to quantitative values for decision is also a big challenge. The accuracy and reliability are the main targets in the transformation. Therefore we define the conversion between the qualitative and quantitative characteristics as a quantitative description process for the complex properties and characteristics in problems. Only when the quantitative description possesses certain precision and credibility, the solving will be meaningful. Similar to the simplification of uncertainty and dynamics in manufacturing system, in this aspect we measure the problem attributes mathematically. To improve the exactness of problem modeling and solving efficiency, the transformation of qualitative and quantitative features is an important issue to be considered in manufacturing optimization.

In general, there are several ways to deal with the transformation. A few typical ones are introduced as follows.

(1) Fuzzy quantification: It means to represent different problem qualitative attributes as fuzzy value according to their levels and intensions.
(2) Functional quantification: This method defines a fitting function in a certain range to describe the attribute variations with time or environment.
(3) Discrete quantification: It refers to describe qualitative features with a set of discrete values in a certain domain. It is not only for discrete attributes, but also for continuous ones as a compromise between fuzzy quantification and functional quantification.
(4) Stochastic quantification: It is especially for uncertain features in problem. The quantitative values can usually obtained by a series of Monte-Carlo or other stochastic tests. It is inaccuracy but can better describe the uncertainty of qualitative features.

The above four methods are provided for different kinds of problems as methods of transformation between qualitative and quantitative attributes. The first three methods can be well applied in engineering, while the fourth one is less applied because its accuracy and reliability are hard to verify. The fourth method is only suitable for a few problems which have extremely uncertain attributes.

In existing studies, the studies targeted on the transformation of qualitative and quantitative attributes are quite few in manufacturing. Most of them do quantitative conversion and problem modeling based on the above methods without consideration of the accuracy and reliability verification of the model. Nevertheless, the accuracy and reliability are usually the deciding thresholds of the quantification. If transformation method is not verified, the model will bring unconvincing decision in engineering applications, or even lead to large deviation to the solutions and cause big loss. Therefore, the verification step in this issue is a more important factor in problem modeling and it is more challenging. In the

studies of optimal decisions related to manufacturing, the transformation of qualitative and quantitative attributes exists widely, such as [88–95].

2.3.6 Simplification of Large-Scale Solution Space

With the complication of manufacturing system and the whole life cycle of production, manufacturing resources and processes are getting to be abundant, and the solution space of the optimization problems is getting bigger. With the increasing of the solution space, the accuracy and the time efficiency of existing algorithms are decreased a lot. Hence, simplification of large-scale solution space is also one of a big challenge to better adapt the optimal searching. Specifically, the simplification of large-scale solution space is a process to divide or simplify the problem and solve it in multi-steps with lower complexity.

Facing with large-scale complex problems, the common methods for the simplification of large-scale solution space are:

(1) Divide and conquer: It means to separate a problem into several sub-problems and narrow the size of sub solution space in each optimization step.
(2) Decrease and conquer: This method tries to find a mapping relation between the original problem and another problem with small solution space. Getting rid of unfeasible solution regions according to the constraints can be also fully applied in this method.
(3) Transform and conquer: By instance simplification, representation change and problem reduction, this method aims to transform the original problem to another representation and reduce the solution space during the process.

The three kinds of methods are originally used as deterministic algorithms for different sorts of optimization. They are also effective in dealing with large-scale solution space. The simplification is generally done by conducting a mapping scheme between the solution spaces of the original problem and the simplified one. In intelligent optimization algorithm, the way to simplify the large-scale solution space is usually the encoding scheme. Now many studies on the simplification of large-scale solution space have shown up. The most prominent and effective studies are divide and conquer and its improvements.

However, from the perspective of engineering solving, the complexity of existing problems is gradually increasing, and there are endless kinds of problems. The simplification analysis of problem solution space requires a lot of time, and the exponential exploration in deterministic optimization is still not well solved. In the solving process of large-scale complex problem, finding a general method to simplify large-scale solution space for various kinds of special complex problems is still a big challenge. For the existing studies in simplification of large-scale solution space, refer to references [96–102].

2.3.7 *Jumping Out of Local Convergence*

According to the above discussion, many deterministic algorithms can not find optimal solution in polynomial time owing to the growing complexity and scale of problems in complex manufacturing system or process. Therefore, various kinds of nondeterministic algorithms such as intelligent optimization algorithms are presented. These algorithms aim at giving feasible sub-optimal solutions of problem in a short time, and conduct stochastic and heuristic search in the solution space. The core issue in nondeterministic algorithm is how to jump out of local convergence and find better sub-optimal solutions.

Jumping out of local convergence refers to design strategies in algorithm which can promote the stochastic evolutionary process to find better solutions in the situation of local convergence. When an algorithm is trapped into local convergence, it will search repetitively in a small region until terminal conditions are reached. Early convergence will definitely lead to low efficiency and high time consuming in problem-solving. In over 30 years of theoretical study, researchers performed in-depth analysis about the convergence of many iterative-based algorithms. However, only a few are verified theoretically so far. From the perspective of practice, researchers made efforts on the design of algorithm improvements to escape from early convergence in solving different problems, such as the increasing of search step, the eliminating of similar solutions, the importing of chaos and the adaptive parameter tuning. Many of them have been applied in various kinds of engineering problems. They have high reusability and have their own focus in specific problem.

However, the strategies for jumping out of local convergence have not been effectively improved. Due to the huge scale solution space, high stochastics, unsuitable heuristics and so on, it is harder and harder to improve the efficiency of problem-solving. There is no free lunch. Facing with expansive complex problems, handling the balance between exploration and exploitation with iterative-based local optimization are discussed a lot. Jumping out of local convergence is still one of the huge challenges in today algorithm design and problem solving. For more instances, refer to the references [103–110].

2.4 An Overview of Optimization Methods in Manufacturing

Facing so many challenges, researchers and engineers keep looking for high efficient optimization method to solve those complex problems in manufacturing system and process. On the whole, we may divide the optimization methods into six categories according to their design and solving process, i.e. Empirical-based method, Tool-based method, Prediction-based method, Simulation-based method, model-based method and Advanced-computing-technology-based method. All

those methods require the support of intelligent optimization algorithm in solving most problems. Therefore, we briefly describe the six kinds of methods in manufacturing and then show the key elements in design with typical examples.

2.4.1 Empirical-Based Method

Empirical-based method generally refers to the optimization according to the reasoning and analysis based on experience information in problem modeling. It is mainly applied in the situation that some properties of the problems, such as the variable domain and range, can not be defined, or problems with stochastic and large solution scale that can not be traversed. Typical instances are in the process control of complex system, and the parameter selection in product design and so forth. Some classical schemes applied in empirical-based methods are as follows.

(1) Empirical local search: It is defined as empirical selecting and narrowing the domain of variables to be solved in the problems according to environmental information, and searching locally in a small solution region. This process is actually an empirical selection of searching domain.
(2) Empirical stochastic search: By dividing the solution-space, it tries to set the search probability and success rate of different solution area according to the experience and information, and obtains the feasible sub-solutions by random searching. This process is an empirical selection of search probability.

These methods mostly use empirical environment data to define the properties of problem and divide and check the searching area so as to simplify the optimization process. There are many studies on the modeling of empirical data or features in complex problems. In empirical-based methods, the key point is that the verification and selection of reliable empirical data and priori knowledge. It is mainly used to deal with the challenges like the modeling of uncertainty and dynamics, and the simplification of large-scale solution space for different manufacturing environments. In the problems which require empirical information, accuracy requirements in optimization are usually low, while the requirements of feasibility and efficiency are high. Therefore, the empirical-based solving is mostly indispensably combined within intelligent optimization algorithms and other uncertainty algorithms like approximation algorithms to design. It is largely problem independent with manual regulation. Besides the cases given in the literature in last section, refer to references [111–117] for the studies in empirical-based solving.

2.4.2 Prediction-Based Method

Prediction-based method generally refers to the optimization in which some problem or algorithm attributes are trained and predicted based on environmental information during the process. The solving process is guided by the changing predicted parameters. It is mainly applied in the situations that the accuracy requirement of decision is high, but the problems are real-time, dynamic and uncertain and the dynamic and uncertain parameters can be modeled with time-stepping iteration. These problems usually appear in dynamic time-based scheduling, and real-time control for manufacturing process and so forth. Thus in manufacturing, prediction-based strategies are used in problems like parameter optimization, combinatorial scheduling, and detection and classification the most. The most applied prediction schemes are listed as follows:

(1) Prediction with fitting function: Based on the change rules of the prior tested data and attributes along with the time and environment advance, this strategy tends to carry out prediction with the fitting function of these rules. In each step of prediction, the problem attributes in the next period are calculated according to the fitting function.
(2) Fuzzy prediction: In this strategy, the attribute values are generally divided into several levels, and the prior tested data are clustered with fuzzy processing. After that, the mapping relation between attribute levels and the changing environment needs to be conducted to guide the prediction.
(3) Prediction with classification: Combining with classification algorithms, this strategy establishes a training model according to the problem priori dataset. On the basis of the training model, the key states of problem in the next period can be predicted for next step decision.

In addition, there are many prediction strategies applied for dynamic optimization. Most of them firstly model and analyze the mapping relation between problem features and environmental dynamics according to priori data, then guide the optimization by predicted model to improve the solving accuracy. They can be widely applied in dealing with challenges like the modeling of uncertainty and dynamics, transformation of qualitative and quantitative features, balance of multi-objectives and handling of multi-constraints. In this method, the key point is the accuracy of the predicted model. Due to the prediction training or mapping and the real-time optimization can be performed in parallel, the requirement on time consumption of prediction are lower. With the increase of the manufacturing and system complexity, the prediction-based method gradually requires the short-time and dynamic invoking of intelligent optimization algorithm to better serve different situation with iterative searching. For the prediction-based method in manufacturing field, refer to references [118–125].

2.4.3 Simulation-Based Method

Simulation-based method generally defines as the optimization which obtains problem states by real-time simulation, and solves the problem by state monitoring with feedback compensation. This method is mainly applied in the situation that the problems are dynamic and have the feature of strong real-time, or the problem attributes and environmental parameters can not be modeled exactly. Furthermore, when the predicted data is hard to obtain or not able to obtain as the time is insufficient, this method is more useful and accurate than the prediction-based method. But sometimes the simulation is hard to implement and its application scope is narrow. It is a mainly applied optimization method in manufacturing for complex product modeling and designing. Thus it often appears in finite element analysis and multi-disciplinary optimization. Here are several simulation-based methods:

(1) Real-time monitoring: It is conducted based on several simulation tests. The outputs of real-time monitor are directly applied as the input of optimization. Then dynamic optimization in real-time can be established.
(2) Multi-run simulation: In this method, the problem parameters are obtained by the record of multi-run simulation. The optimization can be carried out based on either the average value or one stochastic value of the output.
(3) Feedback simulation: It refers to simulate the problem with input and output monitoring, regulate the output status according to the real-time system monitoring, and guide the optimization by both the regulating rules and the simulation output.

In manufacturing systems, the first and the second methods are the most commonly used ones. The methods above all take multiple sets of input and output in the simulation as reference variables or objective values. The key point is not the accuracy of simulation, but the dynamic processing ability of algorithm based on simulation. This method can be widely applied in various manufacturing problems with high real-time and dynamics. However, due to the difficulties in simulation construction, research in simulation-based method in recent years is relatively less. It is more used in the modeling of real-time process or product design and can be combined with intelligent optimization algorithms to construct an adaptive evolution and dynamic feedback to solve complex problems. For simulation-based method, refer to references [126–132].

2.4.4 Model-Based Method

The model-based method is the simplest and most commonly used method, which solve the problem according to the mathematical description of its variables, objectives and constructions. Although most of the application and research in

manufacturing simplify the problems with quantitative mathematical description, owing to the simplification of some key dynamic factors, the decision result of this method is somewhat inaccurate. It is mainly applied in the situation that the properties and features of problems are clear without uncertainty and dynamics, such as in balance of multi-objectives and handling of multi-constraints. The key point of this method is the accuracy and reliability of the mathematical description of the problem. Due to its universal application, we will not repeat it here.

2.4.5 Tool-Based Method

The tool-based method mainly refers to the optimization that dynamically extracts problem features and solves the problem with the assistance or guidance of system or process management tools. In a broad sense, the tool-based method can be defined as a kind of simulation-based method. But narrowly speaking, the tool-based method focuses on the use of assistant tools to obtain features and its guidance on optimization in practical, while the simulation-based method focuses on the modeling of problem features in virtual simulation. Like simulation-based method, the tool-based method is mainly applied in the situation that the problem is dynamic and uncertain with time and its properties are hard to extract quantitatively. Compared to the building process of simulation-based system, because of the tool functions, the tool-based extraction of assistant features with monitoring has a lower difficulty to achieve. Its application scope is also wider. It is mainly applied in manufacturing problems such as finite element analysis, parameter optimization and multi-disciplinary optimization.

To the extraction of problem attributes and environmental characteristics, there are various kinds of existing assistant tools. There are also many studies focus on the algorithm design based on the tool assistance, and trying to integrate those tools to improve the reliability and efficiency of algorithms applied in specific environment. The key point is the collaboration process of inputs and outputs of the tools and the designed algorithms, which is similar to the methods mentioned above. For the tool-based method, refer to references [8, 81, 133, 134, 135, 136].

2.4.6 Advanced-Computing-Technology-Based Method

Advanced-computing-technology-based method refers to conduct optimization by ways of parallel, distributed multi-steps and collaborative computing with the help of existing advanced computing resources. This method is different from previous design of solving methods. It is combined with ideas of problem and algorithm partitioning, designed to use existing advanced technology to allocate sub-problems or sub-optimization tasks in distributed resources and solve them in parallel. This method is mainly applied in large-scale problems and multi-disciplinary

optimization. It has the characteristics of high scalability and efficiency. Because of the high requirements of professional technology, the solving process of specific problems based on this method is quite complex in design, and hard to realize. From the perspective of algorithm design based on existing advanced computing technology, there are several typical ways:

(1) Design based on distributed computing: Perform multi-step decision-making by dividing the problem into several steps and design or call different resource service for the module design of each step.
(2) Design based on parallel computing: Divide the problem into several parallel execution modules that the data or control dependencies among them are the fewest. Then divide the modules into different resources and perform parallel computing.
(3) Design on collaborative computing: Divide the problem into several execution modules which have the hybrid relations of series or parallel. Then solve these modules by different resources and algorithms.

The basic ideas of these solving methods based on advanced computing technology are all dividing the problems and using distributed resources efficiently. In this method, because of the dependencies among the modules in the divided problems or algorithms, we have to consider the time-consuming of task communication. Thus the dividing scheme should be customized according to the environment and complexity of the problem. No matter which technology is used to divide the problems and package the modules, the key is always the partitioning and allocation of the whole optimization. In recently, the problem scales in existing manufacturing systems and processes are increasing gradually, and these problems involve several kinds of multi-disciplinary optimization. Therefore, the studies based on advanced computing technology gained prominence gradually. The solving efficiency brought by those methods can't be ignored. The collaboration and hybridation of various kinds of algorithms for solving different subproblems make the analysis and decision of complex problems easier and the application of distributed computing resources make the whole optimization more efficient. But facing the bottlenecks of the communication costs and the implementation of related technologies, general module partition schemes with less communication becomes another challenge. For advanced-computing-technology-based method, refer to references [137–144].

2.4.7 Summary of Studies on Solving Methods

We found that the algorithms designed for complex manufacturing optimizations are hybrid with different kinds of strategies. Therefore there are no clear boundaries among these methods of algorithm design, like the empirical-based method uses prediction at the same time, and the simulation-based method uses assistant tools in many situations, etc.

From the perspective of the application of different methods, it can be seen that the model-based method is mostly used. No matter in dynamic optimization, black-box-based optimization or monitor-based optimization, objectives functions (or evaluation functions), the abstract variables and parameters and their relations are presented as mathematical models. In actual projects, the empirical-based method is right after the model-based method in the number of application times, and it mostly get the assistance of simulation and tools to optimize. In recent years, there are more studies on the prediction-based method and advanced-computing-technology-based method because the prediction of major cases will make the optimization process more intelligent, and the advanced computing technology can always take the advantages of distributed resources to make the solving faster. In addition, because of the complexity and particularity of manufacturing process, there are few studies on simulation-based method and tool-based method appeared in recent years. On the whole, after sampling of recent literatures, we roughly get the mapping of solving methods and the optimization problems, as shown in Fig. 2.5. **The numerical orders we get from the research literatures on different methods are: model-based method > advanced-computing-technology-based method > prediction-based method > empirical-based method > simulation-based method > tool-based method. And the application numbers of them in actual projects are: empirical-based method > tool-based method > simulation-based method > prediction-based method > advanced-computing-technology-based method**. From the difference between the solving methods used in actual projects and in research, we can find out that we have to consider the integration of tools, simulation, and the existing advanced intelligent technology further to narrow the gap between optimization design in projects and in research.

2.5 Intelligent Optimization Algorithms for Optimization Problems in Manufacturing

In summary, there are varieties of complex problems in manufacturing. Decision and optimization face multiple large challenges simultaneously. To deal with these challenges, a series of solving methods including several problem modeling and algorithm design methods have already been proposed. Throughout the studies on the optimizations in manufacturing field, the improvement and application of intelligent optimization algorithms are one of the major parts. No matter modeling the problems by experience, prediction or tools, or designing algorithms based on simulation, heuristics, or advanced computing technology, intelligent optimization algorithm are widely applied in the optimization process because of its independence, versatility and efficiency. At the same time, with multiple challenges in manufacturing problems, there are a series of studies and improvements carried out

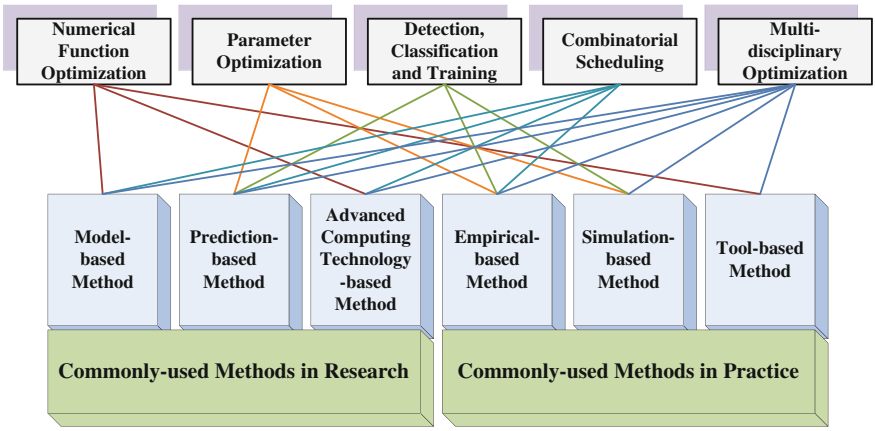


Fig. 2.5 The mapping of the solving methods and the optimization problems in manufacturing

in encoding schemes, operator designs and evolutionary strategies of intelligent optimization algorithms. These achievements are fully reflected in the six sorts of optimization methods

Specifically speaking, in the empirical-based and prediction-based method, intelligent optimization algorithms are required for the selection of empirical or prediction factors and data. In simulation-based and tool-based method, it is needed for adaptive control the modeling and simulation. In advanced-computing-technology-based method, intelligent optimization algorithms are also used to partition the optimization tasks and modules to some extent. On the contrary, all these methods are used and studied as assistant strategies for intelligent optimization algorithms to solve these complex problems with higher efficiency. Facing with the combinatorial explosion generated by complex problems, we can only take advantage of the support of intelligent optimization algorithms with independent iterations and the characteristics of high versatility and scalability to avoid the combinatorial explosion in the problems, and get the satisfying solutions in a short time.

The application review of common intelligent optimization algorithms in manufacturing field can be listed as shown in Table 2.1. It can be clearly seen the research emphasis on different kinds of problems corresponding to different optimization methods in recent years.

Typical cases of each kind of problem in manufacturing field are listed in the table. Each case faces all the seven challenges described before. However, due to the different characteristics of problems, they have different emphasis in dealing with those challenges. In this table, the main challenges to be overcome in these typical cases are shown by check marks. Furthermore, the solving methods mainly taken for different kinds of problems are listed in the next paragraph.

Based on the literature review, it can be found that 60–70 % of the research and applications in solving complex manufacturing problems use different styles of

Table 2.1 Summary of the characteristics, research emphasizes and major methods of complex problems in manufacturing field

Problems	Typical cases in manufacturing	Multi-objectives	Multi-constraints	Priori knowledge	Uncertainty and dynamics	Qualification and quantification	Large-scale solution space	Local convergence	Major methods
Numerical function optimization	Functions in process modeling [2, 9]		✔				✔	✔	Model-based method Advanced-computing-technology-based method Tool-based method
	Finite element analysis [1, 4]		✔				✔	✔	
Parameter optimization	Parameters in control [14, 15]	✔	✔	✔	✔				Empirical-based method Prediction-based method Simulation-based method
	Parameters in machining (Wu et al. [10, 11]	✔	✔	✔					
	Parameters in system [12, 17]	✔	✔	✔					

(continued)

Table 2.1 (continued)

Problems	Typical cases in manufacturing	Multi-objectives	Multi-constraints	Priori knowledge	Uncertainty and dynamics	Qualification and quantification	Large-scale solution space	Local convergence	Major methods
Detection and classification	Detection in machining [28, 29]			✔	✔	✔	✔		Empirical-based method Prediction-based method Simulation-based method
	System training and assessment [23, 24]			✔	✔	✔	✔		
	Classification of model features [21, 27]			✔	✔	✔	✔		
Combinatorial scheduling	Workflow management [31, 35]	✔	✔		✔		✔		Model-based method Prediction-based method Empirical-based method Advanced-computing-technology-based method
	Job shop scheduling [38, 40]	✔	✔		✔		✔		
	Production partner selection [33, 39]	✔	✔		✔		✔		

(continued)

Table 2.1 (continued)

Problems	Typical cases in manufacturing	Multi-objectives	Multi-constraints	Priori knowledge	Uncertainty and dynamics	Qualification and quantification	Large-scale solution space	Local convergence	Major methods
Multi-disciplinary optimization	Collaborative process optimization in manufacturing [41, 43]	✔	✔	✔	✔	✔	✔	✔	Empirical-based method Prediction-based method Simulation-based method Tool-based method Advanced-computing-technology-based method

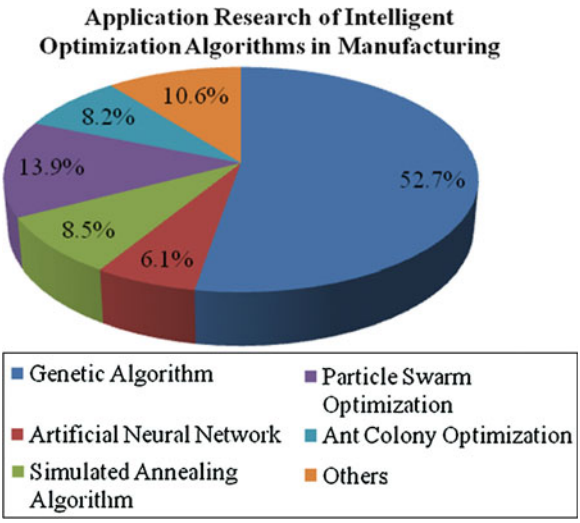


Fig. 2.6 The approximate application distribution of common intelligent algorithms

intelligent optimization algorithms, in which genetic algorithms, artificial neural network algorithms, simulated annealing algorithms, particle swarm algorithms and ant colony algorithms are the most typical and applied. The approximate application distribution of common intelligent optimization algorithms is shown in Fig. 2.6. The most applied one in different kinds of problems is genetic algorithm. Because it is proposed the earliest, and its operators are simple and independent, which means the algorithm is appropriate both for continuous and discrete optimization. However, with the characteristic of premature convergence, different kinds of improvements and combination to the genetic algorithms are designed based on various benchmarks and practical problems. Except genetic algorithm, ant colony algorithm and particle swarm algorithm are applied a lot. The self-learning mechanism of particle swarm algorithm is designed for continuous numerical optimizations. In discrete combinatorial optimization, the updating mechanism of individuals needs to be changed and improved. Most of the improvements transform the original change to the crossover between individual. In this situation, the original particle swarm optimization is transformed as a kind of hybrid genetic particle swarm algorithm to some extent. Overall, particle swarm algorithm is mostly applied in numerical function optimization, parameter optimization and multi-disciplinary optimization. On the contrary, ant colony algorithm is designed for path finding related combinatorial optimization, such as route optimization of robots, task scheduling and so on. In continuous numerical optimization, the searching step size needs to be set beforehand. If the step size is large, the accuracy cannot be guaranteed, if the step size is small, a large-scale pheromone vector is required, which slow down the search. Moreover, ant colony algorithm generally needs the extraction of priori knowledge, which makes the algorithm not very versatile in application. In addition, because typical simulated annealing algorithm and tabu search have strong randomness, and they are carried

out with single individual-based iterations, the probability of getting better solutions in a short time is low. Currently, the hybrid of intelligent optimization algorithms and other algorithms is mostly applied in application, which offers great assistance to improve the searching diversity and guidance.

In addition, there are several typical intelligent optimization algorithms applied in the continuous numerical optimization like simulated-annealing algorithm and differential evolution algorithm, and in the discrete combinatorial optimization like memetic algorithm, ant colony optimization and so on. Other algorithms like immune algorithm, DNA computing algorithm, culture algorithm and newly appeared bee colony algorithm are not mature in development and application. Most of the engineers are not familiar with these new algorithms. As a result, few studies are applied on the problems in manufacturing field. It also reflects an important thing in the study of optimization, that the new better research results are not effectively used in actual projects.

2.6 Challenges of Applying Intelligent Optimization Algorithms in Manufacturing

Currently, intelligent optimization algorithm has become an integral expertise in manufacturing system and process optimization. It helped to breakthrough a lot of difficulties in optimization, like the decision of job shop scheduling and finite element analysis, etc. However, the problem will change with the environment. Therefore, in complex manufacturing systems, to improve the efficiency of problem optimization and decision, the design and development of intelligent optimization algorithm becomes a research hotspot. Although thousands kinds of improvement, hybridation of intelligent optimization algorithms have been proposed, their solving effects on different kinds of specific problems are still unknown. In algorithm design process, different challenges still exist in all of the steps, i.e. problem modeling, algorithm selection, encoding scheming and operator designing. According to the characteristics and requirements of different designing parts, we will analyze the main challenges separately.

2.6.1 Problem Modeling

From the perspective of problem modeling, the three basic elements are variables, objectives and constraints. Modeling of the three elements directly influences the quality of decision. Thus it is the basis of the designing of intelligent optimization algorithm.

Firstly, there is one-to-one relationship between the problem variables and the individuals in algorithm. If the variables are continuous, the factors as domain, search step, accuracy requirements should be explicitly given. If the variables are

discrete, besides the domain, we have to clarify the direct relationships between variables, which will make the following encoding easier.

Secondly, no matter for single-objective or multi-objective problem, the objective functions as evaluation criteria in the algorithm are the essential foundation of search. For the problems can be mathematically modeled, clear objective functions need to be given. For the problems that objective function can not be given, like parameter optimization in process control and detection, we have to test the solution with simulation or monitoring of actual systems to get the result. Then the outputs of the system are taken as the objective values to evaluate the population in algorithm iteration in real-time. It is important to note that too simplified evaluation functions will result in low accurate optimization, while too complex assessment model will lead to large time loss.

Finally, in dealing with constraints, no matter put them in the objective functions as penalty functions or define them as population check during iteration, they will greatly influence the algorithm optimization. Inappropriate handling of the constraints will easily lead to invalid iteration search in unfeasible solution space, which will seriously reduce the efficiency of algorithm.

It is thus clear that in the establishing process of optimization model, the main difficulties are:

(1) The precisely clarification of the properties of problem variables;
(2) The appropriate construction of objective functions (or assessment methods);
(3) The suitable handling way of constraints.

2.6.2 Algorithm Selection

Based on the establishment of problem model, full research and comparison of hundreds of hybrid and improved intelligent optimization algorithms and selection of suitable one for the specific problem is almost impossible. To most engineers, it is even not easy to choose in a set of basic intelligent optimization algorithms. Because almost all of the existing intelligent optimization algorithms have not been verified theoretically, large experiments are mostly required for compare their efficiency to specific problem. Hence, it is quite hard to figure out which algorithm is suitable and which improvement or hybrid strategies can bring about enhancement for specific problems.

Currently, people usually select the most commonly used algorithm according to different application extents of the algorithms and the recommendations from existing research when facing complex problems in manufacturing field. Most of them select the most classic genetic algorithm, and ignore many new intelligent optimization algorithms. Based on the selected algorithm, according to the former procedure, the algorithm is implemented and improved again, which extends the algorithm design cycle and produce a lot of repetitive work. In addition, if the selected algorithm is inappropriate for the problem, such as using a highly

evaluated intelligent optimization algorithm specializing in discrete combinatorial problems to solve specific continuous problem, the algorithm requires to be transformed a lot, and the result of the optimization is still possible to be substandard.

It is thus clear that algorithm selection is the core to decide the optimization efficiency in solving a problem. Currently, there are many integrated libraries which can provide some typical intelligent optimization algorithms. But with different kinds of problems, the merits and demerits of various algorithms can not be compared directly. And also, there are still less study and emphasis on the construction of related algorithm libraries. People are more willing to select mature and convenient method to solve the problem.

Generally speaking, in the process of algorithm selection, the main questions need to be note are as follows:

(1) Less analysis, verification and classification on existing typical intelligent optimization algorithms in solving different problems;
(2) Lacking of unified evaluation methods for various algorithms in solving different problems;
(3) Lacking of a standard integrated algorithm library for algorithm design, comparison and application.

2.6.3 Encoding Scheming

Encoding scheming for problem refers to the process of transforming the key variables into individual genes. It is the band between intelligent optimization algorithm and specific problem combined with fitness function. Population updating in each generation performed by combined operators is also based on encoding scheme.

Normally, binary-coding, real-number-coding and vector-coding are the most commonly used. In some encoding schemes, individuals and variables have one-to-one mapping relationship. However, in most coding form for particular problems, individuals and variables are not one-to-one mapping. When they are having one-to-many relationship, i.e. one individual corresponds to several solutions, the decoding can not be well implemented. When they are having many-to-one relationship, i.e. several individuals correspond to one solution just like the situation of real-number-coding in task scheduling, then invalid searching with repeated iteration and local convergence can easily occur, which is detrimental to the whole evolutionary optimization. Moreover, in combinatorial optimizations like job shop scheduling and traveling salesman problem, the targets are finding the best permutations of variables, which means that the values of different variables can not equal to each other, then the requirement to the coding in such situations is very high. Not only so, operators like crossover in genetic algorithm and self-studying in particle swarm optimization are designed and varied with

different coding scheme. Therefore, coding scheming is crucial in the application of intelligent optimization algorithms.

Currently, the main study aspects and difficulties in encoding scheming for specific complex problems are:

(1) Coping with "many-to-one mapping situation" to avoid repetitive searching.
(2) The avoiding of inflexible solutions with encoding scheming.
(3) The adjustment of encoding scheme for specific operators.

2.6.4 Operator Designing

Based on these three steps of design, people have to improve the algorithm after the selection to adjust the problem and achieve a higher efficiency of optimization. But there are too many operators and improvement strategies. Thus users and designers have to perform further research and analysis on the existing operators and improving strategies based on the encoding scheme and improve the algorithm again. Based on the requirements of the problem and the selected algorithm, the design of improvement strategies for the operators can be seen as a matching combination process. Different type of operators and improvement strategies can form several hybrid and improved algorithms after different permutations and combinations.

In the existing research and application, people usually adjust and combine the operators according to the existing experience, or perform single-step fine tuning in operators and try improved strategies one-by-one to specific problem. The interactions among the operators and the balance between exploration and exploitation in iterations are less considered and analyzed, which leads to great limitations in the performance of existing operators and improvement strategies.

Therefore, in the improvement design process of operators, the main difficulties are:

(1) Lack of analysis in features and combination effect of the operators for different problems.
(2) Lack of balance between exploration and exploitation in algorithm design.
(3) Many good improvement strategies have not been well extended and applied to different types of problems.

2.7 Future Approaches for Manufacturing Optimization

Challenges are not only existed in the above mentioned four steps. With the gradual complication of manufacturing process and system and the proposal of advanced manufacturing model such as networked and service-oriented manufacturing,

the problems in manufacturing field are more evolving into multi-disciplinary large-scale ones. Thus, single deterministic algorithm or intelligent optimization algorithm is far from sufficient to meet the requirement of solving.

From the algorithm designing perspective, it requires the assistances of more than one of the six kinds of solving methods. For the optimization process, different adaptive, exploitation and exploration strategies are needed in different solving stage. For solving the problem in real-time, we need to handle the dynamics and uncertainties with the adaptation and combination of various operators. In a word, the whole solving process for multi-disciplinary complex problems needs multi-level or multi-stage operators combined with multiple decision methods and technologies.

It can be concluded that for the optimization problems in manufacturing field, the requirements are diverse, and the corresponding solving methods have to meet not only the requirements of the system dynamics and uncertainty, but also the ability of efficient collaborative optimization and management. The trend of development can be briefly summarized into several points:

(1) Hybridation of diversified methods.
(2) Multi-stage processing of uncertainties and dynamics.
(3) Technologies for rapid real-time responding and decision-making.

2.8 Future Requirements and Trends of Intelligent Optimization Algorithm in Manufacturing

With such a general trend, intelligent optimization algorithms need further digging to enhance its efficiency, flexibility and scalability according to the requirements of the three main users in manufacturing, i.e. algorithm beginner, algorithm employer and senior researcher, to adapt practical complex decision in manufacturing engineering. Although the theoretical analysis of operators on the intelligent optimization algorithms is of great importance, yet from the aspect of engineering application, implementations of integrated, configurable, parallel and service-oriented intelligent optimization algorithms are becoming the key development trends in solving complex manufacturing problems.

2.8.1 Integration

During the digital industrial producing process, every step in the whole life cycle of manufacturing contains simulation and tool-aided analysis with varieties of professional software. As for every single optimization module in manufacturing, engineers need to implement and encapsulate different kinds of intelligent

optimization algorithms according to I/O interfaces of the module to realize automatic and systematic decision. With the collaboration of diversified tools, integration of intelligent optimization algorithms and other technologies as modules are necessary for implementing high efficient comprehensive decision.

Specifically, on one hand, based on a rich mixture of assistant tools, many environmental parameters and problem attributes can be obtained easier. The support of multiple technologies can be seen as a combination of simulation-based, tool-based and advanced-technology-based methods. Integrating the intelligent optimization algorithms with these assistant tools and technologies can makes them easier to adapt in specific systems and perform better function. On the other hand, integration and encapsulation of multiple intelligent optimization algorithms can make multi-methods' decision possible. For specific problems, engineers can compare different algorithms in practical environment and apply more than one to do optimization in different stages.

On the contrary, if we design and implement the intelligent optimization algorithm in each different environment, the design process of optimization will be more complicated and time consuming. Therefore, in order to achieve simplified and high efficient collaborative optimization, the most convenient way is to integrate diversified intelligent optimization algorithms and multiple technologies together in the form of tools, and provide uniform standard interfaces to connect with different kinds of systems.

In recent years, some research has turned to the integration of basic intelligent algorithms based on the uniform search of population-based iteration. However, most of the existing integrated algorithm platforms or libraries are inapplicable to complex optimization in collaborative manufacturing. They are generally constructed according to traditional continuous numerical benchmarks. The universal connections with different tools or systems are out of consideration. Most of them require the users to familiar with the optimization process of intelligent optimization algorithms and make improvement based on complex program code. The whole design and comparison process are still quite time consumption.

2.8.2 Configuration

On the other hand, though some existing libraries integrate multiple typical intelligent optimization algorithms, they still have difficulty to adapt to the frequent changing manufacturing problems with efficient research ability. Likewise, as for the whole digital manufacturing process, optimization problems exist in every module. But they are quite different with diverse stages and environments. The dynamic adaptation of intelligent optimization algorithms is needed during the procedure. Thus we do not only need the collaborative decision of several algorithms, but also need that the algorithms to be configured dynamically in the process of optimization in manufacturing systems. Not only the parameters should

be configured, but also the operators, improving strategies and the whole algorithm should be adjusted dynamically.

The existing studies consider rarely about the flexible configuration of intelligent optimization algorithm. To the adaptive processing, the studies of recent years focus more on specific problems and design adaptive processing mechanisms or improving strategies in a single algorithm structure. These mechanisms and structures are mostly unchangeable during optimization. When apply such algorithm in dynamic optimization, comprehensive high efficient searching in all stages can not be realized. For changing environments or properties in problem, engineers have to stop the decision process, store the middle data and redesign the algorithm again. It results in not only a loss of time, but also a repeated waste of program code.

Therefore, though intelligent optimization algorithm has the versatility in structure, with specific problems, it still has weaknesses in adaptability and scalability. The existing intelligent optimization algorithm library only code and store various algorithms independently and it is hard to achieve dynamic configuration. To break through the limitations in the collaborative multi-stage optimizations in manufacturing systems, the studies on the dynamic adjustment and configuration of intelligent optimization algorithms and the scalable combination of the algorithms for complex problems are quite important.

2.8.3 Parallelization

With the development of large-scale cluster systems and distributed computing technology, the design of parallelization mode of intelligent optimization algorithm and its application in large-scale projects are extended gradually, and have achieved some effect. From the perspective of algorithm structure, intelligent optimization algorithm generally can performs collaborative search with population provision, thus it has natural parallelism. From the perspective of parallel computing environment, not only the problems can be separated and solve parallel, but also various solution spaces can be searched in parallel. The combination of intelligent optimization algorithm and the parallelized technologies can save much time for the optimization of various complex projects.

Now more and more design and application studies in the parallelization of intelligent optimization algorithm have shown up. Most of them carry out the research from three key elements: parallel topology, individual migration time and number of individuals to be migrated. And the parallelization in intelligent optimization algorithm design is primarily based on population provision, in which the topology is the main consideration. However, different types of parallel intelligent optimization algorithms show different performance in different problems and

environments with the influences of the three key elements. Currently, although there are many parallel intelligent optimization algorithms for specific problems, the scalability and effectiveness are still to be improved and verified.

Therefore, driven by high performance computing technologies, to further improve the solving efficiency of intelligent optimization algorithms in manufacturing, their parallelization design with the consideration of topology and individual migration elements and the extend application of parallel optimization are urgent.

2.8.4 Executing as Service

Similarly, with the spread of service-oriented manufacturing and computing modes, in distributed manufacturing process, the users mostly get the support from invoking the remote services with different functions through network. Thus multiple services invoked by multi-users can realize resource sharing and high efficient collaborative design and production in manufacturing. Now, as the generalization of the concept of service, some simple algorithms have been encapsulated as services and provided in service center. When users are invoking these services, they only need to concern about the inputs and outputs. At the same time, the users expect the transparency of service computing to achieve real-time monitoring, intelligent interruption and dynamically adjustment. From the perspective of the application of intelligent optimization algorithm in manufacturing, the idea of encapsulating these algorithms as services for different users is already realizable. Currently there are some algorithm libraries which provide typical intelligent optimization algorithms in the form of services on the internet for the users to invoke. However, flexibly and efficiently solving complex optimization problems in the manufacturing systems by intelligent optimization algorithms in form of services has not been implemented and there still exist many challenges in improving the performances of algorithm services in wide area network.

Firstly, from the perspective of application, to different complex manufacturing problems, the users not only have to know the characteristics of the encapsulated intelligent optimization algorithms, but also need to combine them flexibly. It also requires the encapsulated algorithms to be highly configurable. Secondly, from the perspective of process, because of the requirements to the transparent service computing, the intelligent optimization algorithms have to be split into operators and encapsulated as fine-grained modules. Moreover, the clear display and control of the population-based iterative evolutionary process are also needed. These key elements rarely studied but crucial to actual projects. Therefore, the adaptability and flexibility design and implementation of service-oriented intelligent optimization algorithms is very imperative.

2.9 Summary

In this chapter, we mainly talked about the development of the application of intelligent optimization algorithms in manufacturing. From the optimization of manufacturing system and process, the problems are divided into numerical function optimization, parameter optimization, detection and classification, combinatorial scheduling and multi-disciplinary optimization according to the characteristics and objectives of the variables. And we summarized the main challenges faced in solving different kinds of problems. From the perspective of optimization ways, we elaborated the six common optimization design methods, namely, model-based method, empirical-based method, prediction-based method, simulation-based method, tool-based method and advanced-computing-technology-based method, and draw the importance of intelligent optimization algorithm in the large-scale complex optimization problems of manufacturing systems. Developed so far, many intelligent optimization algorithms and improved strategies are proposed for specific problems. However, there are still many challenges in the wide application and targeted design of the intelligent optimization algorithms. Thus, we analyzed those challenges one by one, and gave the major trend of studies and development of intelligent optimization algorithms in manufacturing system and process.

After the above analysis, in the studies and development of intelligent optimization algorithms, though the studies of improvement and design from different perspectives produce a large number of intelligent operators and improving strategies, the requirements of the three kinds of users are far from being fully satisfied. Its main problems are:

(1) Lack of uniformly platform for collection and comparison. Though living in a world with abounds of numerous intelligent algorithms, we still have no idea which one is the best for a particular set of problems due to the lack of integrated centers which are capable of performing standard testing and comparing.
(2) Long design and implementing process. Owning to the sophisticated investigating and programming process of searching and implementing new operators, engineers may limit the usage into several basic algorithms. Such inertia is likely to carry some risks since the generally-used algorithms may not fit well to the given problem, and at the same time, those valuable findings may lose the chance of being used.
(3) Lack of extension and much repetition. Though more and more innovative practices have been designed to enhance algorithms' performance for application-specific demands and general benchmarks, most of them still lack effective testing and extended using. Meanwhile, due to universal unawareness of existing resources, repetitive works have been done in the process of developing same or similar algorithm for different problems in different areas, leading to huge resource wastes and time consuming.

(4) Lack of theoretical foundation. As the scale of the problem (i.e. the solution space) increases, the solving accuracy of the problem drops significantly. Because of the inherent randomness of the algorithm and the searching direction far from completely developed, the balance between the algorithm in exploration and exploitation is still hard to handle. And also, there are not many studies on the theory, convergence and time complexity of intelligent optimization algorithms. The efficiency of the algorithm is obtained by a large number of tests, and lack theoretical foundation.

Moreover, as the high performance computing and service-oriented technologies developing fast, and the scale of combinatorial optimization problems in existing industrial application growing rapidly, the improvement and the application of intelligent optimization algorithms still have long way to go. Maximizing the application of the existing intelligent optimization algorithms and the improvements of them in engineering practices is a difficulty to be solved.

Thus the biggest difficulty turned into: How to effectively employ huge amounts of existing intelligent algorithms and their improvements for various types of uses.

To fully exploit the existing intelligent algorithms and quickly obtain flexible synergies and improvements, new dynamic configuration methods for intelligent optimization algorithms (DC-IOA) is proposed in our work. Based on separated operator modules, three-level configurations, i.e. parameter-based configuration, operator-based configuration and algorithm-based configuration are exploited. Various types of algorithms can be collected and well re-produced by not only arbitrarily combining different operator modules, but also arbitrarily splicing multiple algorithms according to the operational generation separation. Specifically, it can solve the above questions mainly from the following two aspects.

(1) In the view of algorithm employment. Informative workflow with operator modules referring to basic and typical algorithms is provided to algorithm beginners. A friendly interface, where parameter setting, customized operator selecting, and dynamic algorithm combining are involved and provided to senior researchers. In the meantime, various existing algorithms and their improving strategies with only configurable parameters are prepared to for algorithm employers with direct use.
(2) In the view of algorithm development. Comparisons among different strategies are given based on some general benchmarks for algorithm beginner. The encapsulated operator modules and customized interfaces to allow imports of the operators or algorithms and then to support further tests are available for senior researchers. Also recommended algorithms with typical portfolios of operators are given according to the type and feature of submitted problem for algorithm employers.

Starting from the second part of this book, we are going to introduce the theory, design process and application of the configuration method for intelligent optimization algorithm in detail.

References

1. Jung DS, Kim CY (2013) Finite element model updating on small-scale bridge model using the hybrid genetic algorithm. Struct Infrastruct Eng 9(5):481–495
2. Keshavarz S, Khoei AR, Molaeinia Z (2013) Genetic algorithm-based numerical optimization of powder compaction process with temperature-dependent cap plasticity model. Int J Adv Manuf Technol 64:1057–1072
3. Miguel LFF, Lopez RH, Miguel LFF (2013) Multimodal size, shape, and topology optimization of truss structures using the firefly algorithm. Adv Eng Softw 56:23–37
4. Herencia JE, Haftka RT, Balabanov V (2013) Structural optimization of composite structures with limited number of element properties. Struct Multi Optim 47(2):233–245
5. Debout P, Chanal H, Duc E (2011) Tool path smoothing of a redundant machine: application to automated fiber placement. Comput Aided Des 43(2):122–132
6. Hur SH (2011) Modeling and control of a plastic film manufacturing web process. IEEE Trans Industr Inf 7(2):171–178
7. Okaeme NA (2013) Hybrid bacterial foraging optimization strategy for automated experimental control design in electrical drives. IEEE Trans Industr Inf 9(2):668–678
8. Kumar KS, Paulraj G (2011) Genetic algorithm based deformation control and clamping force optimization of workpiece fixture system. Int J Prod Res 49(7):1903–1935
9. Chan KY, Kwong CK, Tsim YC (2010) A genetic programming based fuzzy regression approach to modeling manufacturing processes. Int J Prod Res 48(7):1967–1982
10. Wu Q, Gao L, Li X, Zhang C, Rong Y (2013) Applying an electromagnetism-like mechanism algorithm on parameter optimization of a multi-pass milling process. Int J Prod Res 51(6):1777–1788
11. Yin F, Mao H, Hua L (2011) A hybrid of back propagation neural network and genetic algorithm for optimization of injection molding process parameters. Mater Des 32(6):3457–3464
12. Abd-Elazim SM, Ali ES (2013) A hybrid particle swarm optimization and bacterial foraging for optimal power system stabilizers design. Electr Power Energ Syst 46:334–341
13. Yildiz AR (2013) Cuckoo search algorithm for the selection of optimal machining parameters in milling operations. Int J Adv Manuf Technol 64(1–4):55–61
14. Hassan LH, Moghavvemi M, Almurib HAF, Steinmayer O (2013) Application of genetic algorithm in optimization of unified power flow controller parameters and its location in the power system network. Electr Power Energ Syst 46:89–97
15. Chung IY, Liu WX, Cartes DA, Moon SI (2011) Control parameter optimization for multiple distributed generators in a microgrid using particle swarm optimization. Eur Trans Electr Power 21(2):1200–1216
16. Rai JK, Brand D, Slama M, Xirouchakis P (2013) Optimal selection of cutting parameters in multi-tool milling operations using a genetic algorithm. Int J Prod Res 49(10):3045–3068
17. Juang CF, Chang PH (2010) Designing fuzzy-rule-based systems using continuous ant colony optimization. IEEE Trans Fuzzy Syst 18(1):138–149
18. Rao RV, Kalyankar VD (2013) Parameter optimization of modern machining processes using teaching-learning-based optimization algorithm. Eng Appl Artif Intell 26:524–531
19. Mukherjee R, Chakraborty S (2013) Selection of the optimal electrochemical machining process parameters using biogeography-based optimization algorithm. Int J Adv Manuf Technol 64(5–8):781–791
20. Mehrjoo M, Khaji N, Ghafory-Ashtiany M (2013) Application of genetic algorithm in crack detection of beam-like structures using a new cracked euler-bernoulli beam element. Appl Soft Comput 13(2):867–886
21. Pastrana S, Mitrokotsa A, Orfila A, Peris-Lopez P (2012) Evaluation of classification algorithms for intrusion detection in MANETs. Knowl Based Syst 36:217–225

22. Ak R, Li Y, Vitelli V, Zio E, Droguett EL, Jacinto CMC (2013) NSGA-II-trained neural network approach to the estimation of prediction intervals of scale deposition rate in oil & gas equipment. Expert Syst Appl 40(4):1205–1212
23. Kuo RJ, Tseng WL, Tien FC, Liao TW (2012) Application of an artificial immune system-based fuzzy neural network to a RFID-based positioning system. Comput Ind Eng 63(4):943–956
24. Yang CC (2011) Constructing a hybrid kansei engineering system based on multiple affective responses: application to product form design. Comput Ind Eng 60(4):760–768
25. del Castillo-Gomariz R, Garcia-Pedrajas N (2012) Evolutionary response surfaces for classification: an interpretable model. Appl Intell 37(4):463–474
26. Lin HY (2013) Feature selection based on cluster and variability analyses for ordinal multi-class classification problems. Knowl Based Syst 37:94–104
27. Kaufmann P, Glette K, Gruber T, Platzner M, Torresen J, Sick B (2013) Classification of electromyograpic signals: comparing evolvable hardware to conventional classifiers. IEEE Trans Evol Comput 17(1):46–63
28. dos Santos Fonseca WA, Bezerra UH, Nunes MVA, Barros FGN, Moutinho JAP (2013) Simultaneous fault section estimation and protective device failure detection using percentage values of the protective devices alarms. IEEE Trans Power Syst 28(1):170–180
29. Li R, Seckiner SU, He D, Bechhoefer E, Menon P (2012) Gear fault location detection for split torque gearbox using AE sensors. IEEE Tans Syst Man Cybern Part C Appl Rev 42(6):1308–1317
30. Behdad M, Barone LC, Bennamoun M, French T (2012) Nature-inspired techniques in the context of fraud detection. IEEE Trans Syst Man Cybern Part C Appl Rev 42(6):1273–1290
31. Wu J, Zhang WY, Zhang S, Liu YN, Meng XH (2013) A matrix-based Bayesian approach for manufacturing resource allocation planning in supply chain management. Int J Prod Res 51(5):1451–1463
32. Stadtler H, Sahling F (2013) A lot-sizing and scheduling model for multi-stage flow lines with zero lead times. Eur J Oper Res 225(3):404–419
33. Niu SH, Ong SK, Nee AYC (2012) An enhanced ant colony optimizer for multi-attribute partner selection in virtual enterprises. Int J Prod Res 50(8):2286–2303
34. Chen L, Langevin A, Lu Z (2013) Integrated scheduling of crane handling and truck transportation in a maritime container terminal. Eur J Oper Res 225(1):142–152
35. Rafiei H, Rabbani M, Alimardani M (2013) Novel bi-level hierarchical production planning in hybrid MTS/MTO production context. Int J Prod Res 51(5):1331–1346
36. Bortfeldt A, Homberger J (2013) Packing first, routing second—a heuristic for the vehicle routing and loading problem. Comput Oper Res 40:873–885
37. Xing KYY, Han LB, Zhou MC, Wang F (2012) Deadlock-free genetic scheduling algorithm for automated manufacturing systems based on deadlock control policy. IEEE Trans Syst Man Cybern Part B Cybern 42(3):603–615
38. Xu Y, Wang L (2011) Differential evolution algorithm for hybrid flow-shop scheduling problems. J Syst Eng Electron 22(5):794–798
39. Wu C, Barnes D (2010) Formulating partner selection criteria for agile supply chains: a dempster-shafer belief acceptability optimization approach. Int J Prod Econ 125(2):284–293
40. Li X, Zhang Y (2012) Adaptive hybrid algorithms for the sequence-dependent setup time permutation flow shop scheduling problem. IEEE Trans Autom Sci Eng 9(3):578–595
41. Campana EF, Fasano G, Peri D (2012) Penalty function approaches for ship multidisciplinary design optimization (MDO). Eur J Ind Eng 6(6):765–784
42. Farmani MR, Roshanian J, Babaie M, Zadeh PM (2012) Multi-objective collaborative multidisciplinary design optimization using particle swarm techniques and fuzzy decision making. Proc Inst Mech Eng Part C J Mech Eng Sci 226(9):2281–2295
43. Remouchamps A, Bruyneel M, Fleury C, Grihon S (2011) Application of a bi-level scheme including topology optimization to the design of an aircraft pylon. Struct Multi Optim 44(6):739–750

44. Rafique AF, He LS, Zeeshan Q, Kamran A, Nisar K (2011) Multidisciplinary design and optimization of an air launched satellite launch vehicle using a hybrid heuristic search algorithm. Eng Optim 43(3):305–328
45. Wu DD, Zhang YD, Wu DX, Olson DL (2010) Fuzzy multi-objective programming for supplier selection and risk modeling: a possibility approach. Eur J Oper Res 3(1):774–787
46. Ciavotta M, Minella G, Ruiz R (2013) Multi-objective sequence dependent setup times permutation flows hop: a new algorithm and a comprehensive study. Eur J Oper Res 227(2):301–313
47. Li X, Du G (2013) BSTBGA: a hybrid genetic algorithm for constrained multi-objective optimization problems. Comput Oper Res 40(1):282–302
48. Nourmohammadi A, Zandieh M (2011) Assembly line balancing by a new multi-objective differential evolution algorithm based on TOPSIS. Int J Prod Res 49(10):2833–2855
49. Omkar SN, Senthilnath J, Khandelwal R, Narayana Naik GN, Gopalakrishnan S (2011) Artificial bee colony (ABC) for multi-objective design optimization of composite structures. Appl Soft Comput 11(1):489–499
50. Manupati VK, Deo S, Cheikhrouhou N, Tiwari MK (2013) Optimal process plan selection in networked based manufacturing using game-theoretic approach. Int J Prod Res 50(18):5239–5258
51. Arikat J, Farahandi MH, Ahmadizar F (2012) Multi-objective genetic algorithm for cell formation problem considering cellular layout and operations scheduling. Int J Comput Integr Manuf 25(7):625–635
52. Naderi B, Aminnayeri M, Piri M, Hairi Yazdi MH (2012) Multi-objective no-wait flow shop scheduling problems: models and algorithms. Int J Prod Res 50(10):2592–2608
53. Santana-Quintero LV, Hernandez-Diaz AG, Molina J, Coello CAC, Caballero R (2010) DEMORS: a hybrid multi-objective optimization algorithm using differential evolution and rough set theory for constrained problems. Comput Oper Res 37(3):470–480
54. Minella G, Ruiz R, Ciavotta M (2011) Restarted iterated pareto greedy algorithm for multi-objective flow shop scheduling problems. Comput Oper Res 38(11):1521–1533
55. Pishvaee MS, Farahani RZ, Dullaert W (2010) A memetic algorithm for bi-objective integrated forward/reverse logistics network design. Comput Oper Res 37(6):1100–1112
56. Xu G, Yang ZT, Long GD (2012) Multi-objective optimization of MIMO plastic injection molding process conditions based on particle swarm optimization. Int J Manuf Technol 58(5–8):521–531
57. Puisa R, Streckwall H (2011) Prudent constraint-handling technique for multi objective propeller optimization. Optim Eng 12(4):657–680
58. Nguyen TT, Yao X (2012) Continuous dynamic constrained optimization—the challenges. IEEE Trans Evol Comput 16(6):769–786
59. Qu BY, Suganthan PN (2013) Constrained multi-objective optimization algorithm with an ensemble of constraint handling methods. Eng Optim 43(4):403–416
60. Jiang H, Ren ZL, Xuan JF, Wu XD (2013) Extracting elite pairwise constraints for clustering. Neurocomputing 99(1):124–133
61. Kellegoz T, Toklu B (2012) An efficient branch and bound algorithm for assembly line balancing problems with parallel multi-manned workstations. Comput Oper Res 39(12):3344–3360
62. Mallipeddi R, Suganthan PN (2010) Ensemble of constraint handling techniques. IEEE Trans Evol Comput 14(4):561–579
63. da Silva EK, Barbosa HJC, Lemonge ACC (2011) An adaptive constraint handling technique for differential evolution with dynamic use of variants in engineering optimization. Oper Eng 12(1–2):31–54
64. Wu CY, Ku CC, Pai HY (2011) Injection molding optimization with weld line design constraint using distributed multi-population genetic algorithm. Int J Adv Manuf Technol 52(1–4):131–141
65. Pan QK, Ruiz R (2013) A comprehensive review and evaluation of permutation flow shop heuristics to minimize flow time. Comput Oper Res 40(1):117–128

66. Prakash A, Chan FTS, Deshmukh SG (2011) FMS scheduling with knowledge based genetic algorithm approach. Expert Syst Appl 38(4):3161–3171
67. Xin B, Chen J, Peng ZH, Dou LH, Zhang J (2011) An efficient rule-based constructive heuristic to solve dynamic weapon target assignment problem. IEEE Trans Syst Man Cybern Part A Syst Humans 41(3):598–606
68. Prakash A, Chan FTS, Deshmukh SG (2012) Application of knowledge-based artificial immune system (KBAIS) for computer aided process planning in CIM context. Int J Prod Res 50(18):4937–4954
69. Xing LN, Chen YW, Wang P, Zhao QS, Xiong J (2010) Knowledge-based ant colony optimization for flexible job shop scheduling problems. Appl Soft Comput 10(3):888–896
70. Hsu YY, Tai PH, Wang MW, Chen WC (2011) A knowledge-based engineering system for assembly sequence planning. Int J Adv Manuf Technol 55(5–8):763–782
71. Li BM, Xie SQ, Xu X (2011) Recent development of knowledge-based systems, methods and tools for one-of-a-kind production. Knowl Based Syst 24(7):1108–1119
72. Yu ZW, Wong HS, Wang DW (2011) Neighborhood knowledge-based evolutionary algorithm for multi objective optimization problems. IEEE Trans Evol Comput 15(6):812–831
73. Karimi H, Rahmati SHA, Zandieh M (2012) An efficient knowledge-based algorithm for the flexible job shop scheduling problem. Knowl Based Syst 36:236–244
74. Weiss D (2010) Feature-based spline optimization in CAD. Struct Multi Optim 42(4):619–631
75. Sandberg M, Tyapin L, Kokkolaras M, Isaksson O, Aidanpaa JO, Larsson T (2011) A knowledge-based master-model approach with application to rotating machinery design. Concurrent Eng Res Appl 19(4):295–305
76. Bui LT, Michalewicz Z, Parkinson E, Abello MB (2012) Adaptation in dynamic environments: a case study in mission planning. IEEE Trans Evol Comput 16(2):190–209
77. Zhang XM, Zhu LM, Zhang D, Ding H, Xiong YL (2012) Numerical robust optimization of spindle speed for milling process with uncertainties. Int J Mach Tools Manuf 61:9–19
78. Bekker J, Aldrich C (2011) The cross-entropy method in multi-objective optimization: an assessment. Eur J Oper Res 211(1):112–121
79. Van Hentenryck P, Bent R, Upfal E (2010) Online stochastic optimization under time constraints. Ann Oper Res 177(1):151–183
80. Shnits B (2012) Multi-criteria optimization-based dynamic scheduling for controlling FMS. Int J Prod Res 50(21):6111–6121
81. Dou JP, Wang XS, Wang L (2012) Machining fixture layout optimization under dynamic conditions based on evolutionary techniques. Int J Prod Res 50(15):4294–4315
82. Lee DK, Starossek U, Shin SM (2010) Topological optimized design considering dynamic problem with non-stochastic structural uncertainty. Struct Eng Mech 36(1):79–94
83. Pillac V, Gendreau M, Gueret C, Medaglia AL (2013) A review of dynamic vehicle routing problems. Eur J Oper Res 225(1):1–11
84. Siddiqui S, Azarm S, Gabriel S (2011) A modified benders decomposition method for efficient robust optimization under interval uncertainty. Struct Multi Optim 44(2):259–275
85. Zhang Y, Gong DW, Zhang JH (2013) Robot path planning in uncertain environment using multi objective particle swarm optimization. Neurocomputing 103:172–185
86. Pan F, Nagi R (2010) Robust supply chain design under uncertain demand in agile manufacturing. Comput Oper Res 37(4):668–683
87. Aghezzaf EH, Sitompul C, Najid NM (2010) Models for robust tactical planning in multi-stage production systems with uncertain demands. Comput Oper Res 37(5):880–889
88. Sivakumar K, Balaurugan C, Ramabalan S (2012) Evolutionary multi-objective concurrent maximization of process tolerances. Int J Prod Res 50(12):3172–3191
89. Zhao P, Zhou HM, Li Y, Li DQ (2010) Process parameters optimization of injection modeling using a fast strip analysis as a surrogate model. Int J Adv Manuf Technol 49(9–12):949–959

90. Duffuaa SO, El-Ga'aly A (2013) A multi-objective optimization model for process targeting process sampling plans. Comput Ind Eng 64(1):309–317
91. Lau HCW, Jiang ZZ, Ip WH, Wang DW (2010) A credibility-based fuzzy location model with hurwicz criteria for the design of distribution systems in B2C e-commerce. Comput Ind Eng 59(4):873–886
92. Laili YJ, Tao F, Zhang L, Sarker BR (2012) A study of optimal allocation of computing resources in cloud manufacturing systems. Int J Adv Manuf Technol 63(5–8):671–690
93. Hwang R, Katayama H (2010) Integrated procedure of balancing and sequencing for mixed-model assembly lines: a multi-objective evolutionary approach 48(21):6417–6441
94. Jenab K, Ahi P (2010) Fuzzy quality feature monitoring model. Int J Prod Res 48(17):5021–5030
95. Luo GH, Noble JS (2012) An integrated model for cross dock operations including staging. Int J Prod Res 50(9):2451–2464
96. Kang M, Yoon K (2011) An improved best-first branch-and-bound algorithm for unconstrained two-dimensional cutting problems. Int J Prod Res 49(15):4437–4455
97. Almoustafa S, Hanafi S, Mladenovic N (2013) New exact method for large asymmetric distance-constrained vehicle routing problem. Eur J Oper Res 226(3):386–394
98. Letchford AM, Miller SJ (2012) Fast bounding procedures for large instances of the simple plant location problem. Comput Oper Res 39(5):985–990
99. Sabouni MTY, Logendran R (2013) A single machine carryover sequence-dependent group scheduling in PCB manufacturing. Comput Oper Res 40(1):236–247
100. Karsu O, Azizoglu M (2012) The multi-resource agent bottleneck generalized assignment problem. Int J Prod Res 50(2):309–324
101. Liu Q, Yang XD, Jing L, Li J, Li J (2012) A parallel scheduling algorithm for reinforcement learning in large state space. Front Comput Sci 6(6):631–646
102. Zhu J, Li XP, Shen WM (2012) A divide and conquer-based greedy search for two-machine no-wait job shop problems with makespan minimization. Int J Prod Res 50(10):2692–2704
103. Zhang HF, Zhou JZ, Fang N, Zhang R, Zhang YC (2013) An efficient multi-objective adaptive differential evolution with chaotic neuron network and its application on long-term hydropower operation with considering ecological environment problem. Int J Electr Power Energ 45(1):60–70
104. Alam MS, Islam MM, Yao X, Murase K (2012) Diversity guided evolutionary programming: a novel approach for continuous optimization. Appl Soft Comput 12(6):1693–1707
105. Perez E, Posada M, Herrera F (2012) Analysis of new niching genetic algorithms for finding multiple solutions in the job shop scheduling. J Intell Manuf 22(3):341–356
106. Kaveh A, Zolghadr A (2012) Truss optimization with natural frequency constraints using a hybridized CSS-BBBC algorithm with trap recognition capability. Comput Struct 102:14–27
107. Boussaid I, Chatterjee A, Siarry P, Ahmed-Nacer M (2011) Two-stage update biogeography-based optimization using differential evolution algorithm (DBBO). Comput Oper Res 38(8):1188–1198
108. Chang PC, Huang WH, Ting CJ (2010) Dynamic diversity control in genetic algorithm for mining unsearched solution space in TSP problems. Expert Syst Appl 37(3):1863–1878
109. Chowdhury S, Tong WY, Messac A, Zhang J (2013) A mixed-discrete particle swarm optimization algorithm with explicit diversity-preservation. Struct Multi Optim 47(3):367–388
110. Tao F, Zhao DM, Hu YF, Zhou ZD (2010) Correlation-aware resource service composition and optimal-selection in manufacturing grid. Eur J Oper Res 201(1):129–143
111. Soulier B, Boucard PA (2013) A multiparametric strategy for the two step optimization of structural assemblies. Struct Multi Optim 47(4):539–553
112. Sakata S, Ashida F, Tanaka H (2010) Stabilization of parameter estimation for kriging-based approximation with empirical semivariogram. Comput Methods Appl Mech Eng 199(25–28):1710–1721

113. Bozca M, Fietkau P (2010) Empirical model based optimization of gearbox geometric design parameters to reduce rattle noise in an automotive transmission. Mech Mach Theory 45(11):1599–1612
114. Kondayya D, Krishna AG (2013) An integrated evolutionary approach for modeling and optimization of laser beam cutting process. Int J Adv Manuf Technol 65(1–4):259–274
115. Paralikas J, Salonitis K, Chryssolouris G (2010) Optimization of roll forming process parameters—a semi-empirical approach. Int J Adv Manuf Technol 47(9–12):1041–1052
116. Raja SB, Baskar N (2011) Particle swarm optimization technique for determining optimal machining parameters of different work piece materials in turning operation. Int J Adv Manuf Technol 54(5–8):445–463
117. Ding TC, Zhang S, Wang YW, Zhu XL (2010) Empirical models and optimal cutting parameters for cutting forces and surface roughness in hard milling of AISI H13 steel. Int J Adv Manuf Technol 51(1–4):45–55
118. Chou CJ, Chen LF (2012) Combining neural networks and genetic algorithms for optimizing the parameter design of the inter-metal dielectric process. Int J Prod Res 50(7):1905–1916
119. Sedighi M, Afshari D (2010) Creep feed grinding optimization by an integrated GA-NN system. J Intell Manuf 21(6):657–663
120. Lou P, Liu Q, Zhou ZD, Wang HQ, Sun SX (2012) Multi-agent-based proactive-reactive scheduling for a job shop. Int J Adv Manuf Technol 59(1–4):311–324
121. Hsieh L, Chen WS, Liu CH (2011) Motion estimation using two-stage predictive search algorithms based on joint spatio-temporal correlation information. Expert Syst Appl 38(9):11608–11623
122. Norouzi A, Hamedi M, Adineh VR (2012) Strength modeling and optimizing ultrasonic welded parts of ABS-PMMA using artificial intelligence methods. Int J Adv Manuf Technol 61(1–4):135–147
123. Zhu HP, Liu FM, Shao XY, Zhang GJ (2010) Integration of rough set and neural network ensemble to predict the configuration performance of a modular product family. Int J Prod Res 48(24):7371–7393
124. Jeung HS, Choi HG (2012) Particle swarm optimization in multi-stage operations for operation sequence and DT allocation. Comput Ind Eng 62(2):442–450
125. Chan KY, Kwong CK, Tsim YC (2010) Modelling and optimization of fluid dispensing for electronic packaging using neural fuzzy networks and genetic algorithms. Eng Appl Artif Intell 23(1):18–26
126. Guan YJ, Yuan GP, Sun S, Zhao GQ (2013) Process simulation and optimization of laser tube bending. Int J Adv Manuf Technol 65(1–4):333–342
127. Pirard F, Iassinovski S, Riane F (2011) A simulation based approach for supply network control. Int J Prod Res 49(24):7205–7226
128. Azadeh A, Negahban A, Moghaddam M (2012) A hybrid computer simulation-artificial neural network algorithm for optimization of dispatching rule selection in stochastic job shop scheduling problems. Int J Prod Res 50(2):551–566
129. Varthanan P, Murugan N, Kumar G, Parameswaran S (2012) Development of simulation-based AHP-DPSO algorithm for generating multi-criteria production-distribution plan. Int J Adv Manuf Technol 60(1–4):373–396
130. Varthanan PA, Murugan N, Kumar GM (2012) A simulation based heuristic discrete particle swarm algorithm for generating integrated production-distribution plan. Appl Soft Comput 12(9):3034–3050
131. Azadeh A, Moghaddam M, Geranmayeh P, Naghavi A (2010) A flexible artificial neural network—fuzzy simulation algorithm for scheduling a flow shop with multiple processors. Int J Adv Manuf Technol 50(5–8):699–715
132. Zhang Y, Agogino AM (2012) Hybrid evolutionary optimal MEMS design. Int J Adv Manuf Technol 63(1–4):305–317
133. Kafashi S (2011) Integrated setup planning and operation sequencing (ISOS) using genetic algorithm. Int J Adv Manuf Technol 56(5–8):589–600

134. Vishnupriyan S, Majumder MC, Ramachandran KP (2011) Optimal fixture parameters considering locator errors. Int J Prod Res 49(21):6343–6361
135. Kumar KS, Paulraj G (2012) Geometric error control of workpiece during drilling through optimization of fixture parameter using a genetic algorithm. Int J Prod Res 50(12):3450–3469
136. Huang X, Xie YM, Jia B, Li Q, Zhou SW (2012) Evolutionary topology optimization of periodic composites for extremal magnetic permeability and electrical permittivity. Struct Multi Optim 46(3):385–398
137. Tsai CC, Huang HC, Chan CK (2011a) Parallel elite genetic algorithm and its application to global path planning for autonomous robot navigation. IEEE Trans Industr Electron 58(10):4813–4821
138. Tsai CC, Huang HC, Lin SC (2011b) FPGA-based parallel DNA algorithm for optimal configurations of an omnidirectional mobile service robot performing fire extinguishment. IEEE Trans Industr Electron 58(3):1016–1026
139. Bozejko W, Uchronski W, Wodecki M (2010) Parallel hybrid metaheuristics for the flexible job shop problem. Comput Ind Eng 59(2):323–333
140. Goncalves JF, Resende MGC (2012) A parallel multi-population biased random-key genetic algorithm for a container loading problem. Comput Oper Res 39(2):179–190
141. Jin JY, Crainic TG, Lokketangen A (2012) A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems. Eur J Oper Res 222(3):441–451
142. Boyer V, El Baz D, Elkihel M (2012) Solving knapsack problems on GPU. Comput Oper Res 39(1):42–47
143. Ostrosi E, Fougeres AJ, Ferney M, Klein D (2012) A fuzzy configuration multi-agent approach for product family modeling in conceptual design. J Intell Manuf 23(6):2565–2586
144. Misra R, Mandal C (2010) Minimum connected dominating set using a collaborative cover heuristic for ad hoc sensor networks. IEEE Trans Parallel Distrib Syst 21(3):292–302

Part II
Design and Implementation

Dynamic configuration of intelligent optimization algorithm (DC-IOA) mainly aims to combine existing algorithms and their parts for different complex problems. With the requirement of integration, intelligence and service-orientation in large-scale optimization, we broadly define the configuration of intelligent optimization algorithm as an design process and adaptable idea, in which engineers and algorithm users can on demand change various parameters, select and improve operators belong to different algorithms and hybrid any of them not only before optimization, but also in the running process. Considering the general population-based iterative process in intelligent optimization algorithm, the configuration is divided into three layers, parameter-based configuration, operator-based configuration and algorithm-based configuration. Through parameter-based configuration, the potential of one operator for various problems is dig. In operator-based configuration, operators can be changed in different optimization stages and combined together for extensive use. As a result, more hybrid algorithms with existing operators can be generated. With algorithm-based configuration, the iterative process can be divided into several parts and multiple algorithms can be applied simultaneously or in different steps. Sharing the same population, the balance of exploration and exploitation in iterative optimization is easily obtained. Such idea can not only be used to generate improved and hybrid intelligent optimization algorithms, but also be adopted to design parallel algorithms in some specific parallel architecture.

Therefore, in the Part II of this book, the detail of dynamic configurable intelligent optimization is introduced. This part contains Chaps. 3–5. Chapter 3 presents the basic concept, principle and framework of dynamic configuration method. According to the basic idea, Chap. 4 gives the details of the method for improvement and hybridization of intelligent optimization algorithms. The classifications of improvement and hybridization of intelligent optimization algorithm are involved. More specific implementation ways of dynamic configuration are

established. Further, considering the miniaturization and lighting demand of optimization, we elaborate different kinds of parallel implementation methods of intelligent optimization algorithm and extensively design some new parallelization ways for algorithm with dynamic configuration.

Chapter 3
Dynamic Configuration of Intelligent Optimization Algorithms

Since genetic algorithm (GA) presented decades ago, large amount of intelligent optimization algorithms and their improvements and mixtures have been putting forward one after another. However, little works have been done to extend their applications and verify their competence in different problems. For each specific complex problem, people always take a long time to find appropriate intelligent optimization algorithm and develop improvements. To overcome these shortcomings, new dynamic configuration methods for intelligent optimization algorithms (DC-IOA) [1] is presented in this paper on the basis of the requirements of three kinds of algorithm users. It separates the optimization problems and intelligent optimization algorithms, modularizes each step of algorithms and extracts their core operators. Based on the coarse-grained operator modules, three-layer dynamical configurations, i.e. parameter-based configuration, operator-based configuration and algorithm-based configuration, are fully exploited and implemented. Under these methods, dozens of hybrid and improved intelligent optimization algorithms can be easily produced in a few minutes just based on several configurable operator modules. Also, problem-oriented customizations in configurations can further extend the application range and advance the efficiency of the existing operators enormously. Experiments based on the established configuration platform verify the new configuration ways of applying and improving intelligent optimization algorithm for both numerical and combinatorial optimization problems in industries on aspects of flexibility, robustness, and reusability.

3.1 Concept and Mainframe of DC-IOA

In this section, we propose the mainframe, process and implementation of DC-IOA [1].

As we know, researchers have already developed many algorithm libraries for different use. HeuristicLab is an integrated platform for evolutionary algorithm, and EvA2 is a lib for general optimal algorithms.

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_3

DC-IOA is not only an algorithm library, but also a kind method for algorithm design. Firstly, HeuristicLab and EvA2 packed the algorithms as a whole. However in DC-IOA, algorithms are split into operators and packed as different operator modules. It is a kind of fine-grained algorithm encapsulation. Secondly, DC-IOA allows multi algorithms used in one problem with time sharing. Thirdly, parameters, operators and algorithms in problem solving process can be dynamically configured in DC-IOA. These are the main differences between DC-IOA and other algorithm library.

Hence, the primary goals are concisely clarified as below:

- To make full use of various in-stock algorithms and strategies and realize multi-level of configuration agility;
- To shorten designing and implementing process for mass complicated optimizations;
- To serve as an integrated platform of intelligent optimization algorithms for learning and standard testing purpose.

3.1.1 Mainframe of DC-IOA

The mainframe of DC-IOA can be depicted in Fig. 3.1, each module is connected with each other based on evolutionary population.

Firstly, problem specification mainly addresses the issue of what to optimize. The depiction of problem should include objectives, variables and their upper and lower bounds. The restriction of the problem can be involved either in the fitness functions or in the bounds of the variables. By selecting specific encoding scheme, the population can be generated with different gene values corresponding to the

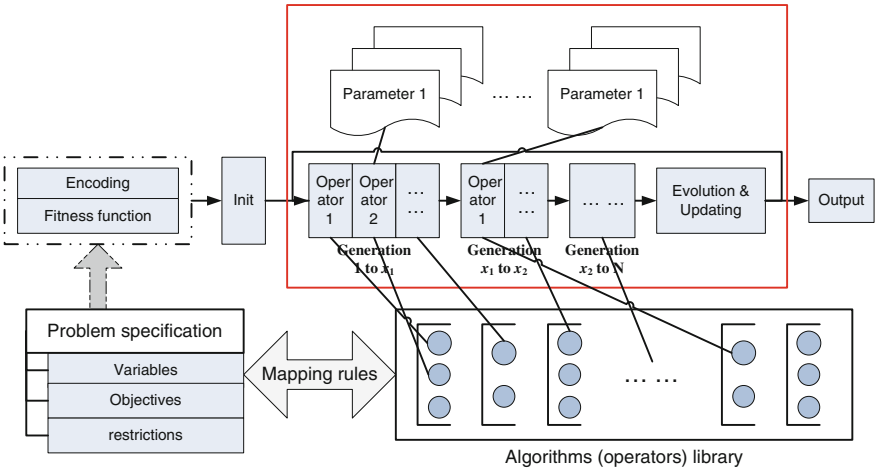


Fig. 3.1 Mainframe of DC-IOA

variables. Then, in accordance with the encoding method and fitness function, we could adopt an initial way to locate the individuals in different positions. After that, problem definition step is finished.

Secondly, we can divide N generations into several segments. On the basis of the operator pool in library, multiple algorithms or hybrid operators can be adopted in different segment with start generation and end generation. Moreover, by dividing population into multiple groups, the algorithms can also be applied simultaneously in different groups to realize algorithm-based configuration. During the process, if the population is trapped into local convergence, then we can replace the operators in the next segment with large mutation operator or other exploration operators to reallocate most of the individuals and simultaneously keep some good information obtained from the last segment.

In each segment, we can either adopt several operators or their modifications stored in the library to formulate an intelligent optimization algorithm, or adopt only one for only exploration or exploitation. In the previous case, complete algorithms need to be designed based on existing operators. The time complexity will be large but good solution can generally be found in earlier stages. Besides, we can also adopt the operators with high exploration in the first stages and apply the ones with high exploitation in the last stages. It is easy to implement and the time complexity in this case will be small. However, good solutions in the earlier stages are relatively hard to find.

Moreover, in each operator, the relative parameters can be encapsulated as different interfaces to implement parameter-based configuration in different searching stages. According to the generation division, we can also apply same algorithm in all these segments and tune the control parameters in different stages.

With the separation of generation and operator collaboration in such framework, the balance of exploration and exploitation is easier to be controlled especially in large scale solution space. We can dynamically generate various intelligent optimization algorithms with existing operators in algorithm library for adapting various kinds of problems.

In addition, for different users, some mapping rules can also be introduced for recording the performance of different algorithms and operator portfolios for different class of problems and recommending suitable operators, algorithms and portfolios to different types of problems by some learning schemes.

Through these modules, the entire optimization process can be dynamically configured.

3.1.2 Problem Specification and Construction of Algorithm Library in DC-IOA

As we introduced in the above section, a problem can be depicted by three elements: variables, variable bounds and objectives. Variables can be transferred into genes according to some encoding scheme while objectives can be converted as fitness

functions. If only one fitness function with weighted objectives is provided, then general population updating step can be directly adopted. If multiple fitness functions which represent different objectives are given, the population updating step will be replaced by Pareto strategy which sort the current population and the new population into multi-level front-sets and select better individuals to the next generation.

What's more, in some complex cases, we may need special encoding scheme to represent different variables. If existing encoding scheme cannot fulfill the optimization requirement, we must design a new one with both encoding and decoding modules. These two modules are then invoked in not only the initial stage, but also during the whole generation together with the operators from the library.

As for the algorithms library, it is actually a central information desk, where various algorithms or improved strategies proposed by different scholars are expected to be found. Each basic algorithm is formed by a number of key operators which is represented as blue node in Fig. 3.1. Because the operators are independent based on population, either single operator in algorithm or the entire algorithm can be invoked for problem. If the problem is extracted as coding function and objective functions, operators selected from the algorithms library can be quickly merged into an algorithm for addressing it. One should be noticed that although these operators are act on population and can be freely put in different algorithm segments, they should be invoked according to specific encoding scheme. For example, if general one-point crossover operator is applied for permutation coding, then illegal individuals will always be generated and the whole searching process will be failure. In this case, only exchange sequence-based crossover or other permutation-based operators can be adopted.

In the three-layer configuration structure, we can perform any layer first according to the demand either in the design stage or during the searching process.

(1) Algorithm-based configuration

We already know that different algorithms may possess different capability. Some may focus on exploration while others are expert in exploitation. If we design hybrid algorithms with the whole generation traditionally, it is hard to get better solutions with the balance of capabilities. Thus we presented a new way, i.e., algorithm-based configuration, to do hybridization with full use of existing algorithms. Break through the traditional hybrid way, we separate N generation to X segments with arbitrary lengths. Segments in the top can adopt algorithms with strong exploration capability and less exploitation, segments in the middle should adopt algorithms with balanced capability between exploration and exploitation, and segments in the rear can adopt either algorithms with strong exploitation or still algorithms with balanced capability. It makes the searching process of intelligent optimization algorithm more dynamic and easier to control.

Moreover, with this new configuration way, when the population is evolving toward a local optimal point or is suffering from a low convergent rate or even divergence, the evolving status observed allows us to suspend the process at any time, during which we can change algorithms in the latter segments to avoid undesirable outcomes. By enclosing the whole algorithm as a unit, several different

algorithm units can be employed to make full use of their respect advantages for different problems. In the case that an inefficient algorithm is configured in some segment, other algorithms with suitable operators can still hold the performance and find better solution.

In addition, generation division and population division can be applied at the same time. In this scenario, more algorithms or single operators can be put in different sub-populations and different segments. For a wide range of problems, there are always one or more configured algorithms or operators performs good, then the robustness and the searching capability of the whole searching process for different problems and for some dynamic problems with changing characteristics can be enhanced.

(2) Operator-based configuration

The operator-based configuration in each algorithm segment or each sub-population mentioned above is another important one in our configuration system. Since most improved or hybrid algorithms have their own operators borrowed or newly designed, the mainframe of the algorithm may remain unchanged. If we uniform the input and output of all operators to be population, then the operator-based configuration can be easily implemented.

For example, when selecting the learning operator in particle swarm optimization and the crossover operator in genetic algorithm, a new algorithm, defined as PSO-GA, can be quickly shaped on the condition of the invariance of initialization, encoding and updating. Sometimes, single operators like chaotic operators can also be applied independently to randomly traverse the whole solution spaces at the first stages. It makes the structure of each single algorithm more dynamic and flexible. And with limited existing operators, more hybrid algorithms can be directly generated by operator composition. Even users who do not understand intelligent optimization algorithm well can design hybrid algorithms for their problem after multiple trials.

But what if we wish to add a new operator or strategy, even a whole algorithm into the existing platform? To make the whole configuration process more expandable, there are two ways to obtain newly designed operators or algorithms: one is to import the newly encapsulated modules in the way of dynamic link library, and the other is to run corresponding code through external compilers based on a general operator module template. The former one, though with a high simplicity, is not conductive for synthesis and storage of operators, whereas the latter one is just the reverse. Therefore, a better solution might be a marriage of the two techniques, so that operators designed by users can be embedded either for once or for permanent usage without causing too much effort.

In a word, from the perspective of basement design, only one-time coding for an operator will facilitate multiple deployments within different algorithm architectures. Also, in the application viewpoint, no complex manipulations are required other than checking or dragging modules, which means even those without much professional knowledge might become potential users. What is more, the expandable feature of the platform makes it more convenient for senior researchers to update libraries and perform more flexible and up-to-date testing.

(3) Parameter-based configuration

The parameter-based module in this paper surpasses the traditional parameter setting, which can only alter the value of some factors in a static manner. Parameter-based configuration defines a more extensive concept for parameter configuration, involving dynamically adding parameters, self-adapting strategies, and modifying parameters during iterations.

Since most intelligent optimization algorithms are still lack of systematic and methodical theory support, some researchers commit themselves to experimental testing to obtain the optimal parameters, and others are engaged on developing self-adaption rules. Parameter interfaces of operator modules allowing multiple setting and modifying of parameters facilitate the first type of study. Yet because of its fixed pattern, it becomes futile when we need to add or delete a particular parameter, or change its attributes. To overcome the above limitations and realize dynamic configurations, we could create new modules with different parameter patterns according to function templates and import them through dynamic link library. It enables users to create their own costumed interface functions based on the available analogues in the library, and upon it, the attributes and boundaries of parameters can be altered in simplified manner, parameter control can be easier.

Additionally, being aware of evolutionary status during iterations is needed. This allows a timely fault detection and modification to get a more efficient and reliable optimization.

(4) Mapping rules legislation

The mapping rules are used to recommend modules for addressing submitted optimization problems, to bring convenience especially for those who possess rare professional knowledge about intelligent optimization algorithms in the library. It is defined based on two points of view, classification and performance (i.e. Quality of services).

First of all, operators in algorithm library are generally classified into three categories when they are encapsulated and loaded in library, i.e.:

- better-in-local-searching,
- better-in-global-searching,
- better-in-self-adaptive- searching.

According to the number of divided generation segments, operators which are better in global searching are tend to be recommended in the former segments, operators which are better in local searching are apt to be recommended in the latter segments and operators which are better in self adaptation are inclined to be put in the middle position. The classification can be modeled by general fuzzy logic as shown in Fig. 3.2.

Except that, recommendation should be given largely in accordance with operators' performance either in cooperation works or in combination works. Cooperation work means operators in the same generation segments while combination work prefers as in different generation segments. In both kinds of works, the mainly

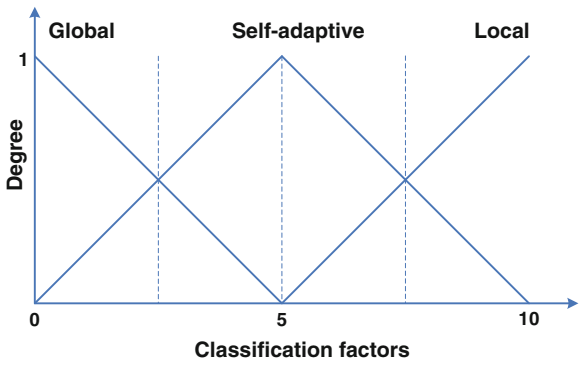


Fig. 3.2 Means of the linguistic classifications associated with fuzzy logic

considered performance properties when an operator portfolio is executed for a specific problem are:

- *BF*: the best fitness f_{best},
- *AF*: the average fitness $\bar{f}$,
- *WF*: the worst fitness f_{worst},
- *VN*: the varience of fitness values, i.e. $(f_{\max} - \bar{f})/(f_{\max} - f_{\min})$,
- *T*: the average execution time,
- *D*: the deviation of the average fitness.

The above properties are generally obtained from multi runs and all of them can be normalized in the range [0, 1] along with changing problems.

For two operators A and B, let the self-performances of them to be $Q(A) = \{s_{a1}, s_{a2}, \cdots, s_{an}\}$ and $Q(B) = \{s_{b1}, s_{b2}, \cdots, s_{bn}\}$ respectively. Let their cooperation performance to be $Q_{coo}(A, B) = \{q_1, q_2, \cdots, q_n\}$ and the combination performance to be $Q_{com}(A, B) = \{p_1, p_2, \cdots, p_n\}$, then the probabilities of recommending A and B in cooperation or in combination can be defined as:

$$P_{self}(A) = \sum\nolimits_n r_i s_{ai}, \quad P_{self}(B) = \sum\nolimits_n r_i s_{bi} \tag{3.1}$$

$$P_{coo} = \sum\nolimits_n w_i q_i \tag{3.2}$$

$$P_{com} = \sum\nolimits_n v_i p_i \tag{3.3}$$

where r_i, w_i and v_i are the weights of different properties in cooperation and combination respectively, and all these weights satisfy the equation $\sum_n r_i = 1$, $\sum_n w_i = 1$ and $\sum_n v_i = 1$.

Therefore, when a problem submitted to the library, operators within different classification with higher P_{self} are firstly selected for different generation segments. Based on the selected operators, new operators which have the highest probabilities in cooperation and combination with the selected ones are then recommended for different segments to produce a portfolio. Note that in each segment, no more

than 4 operators are selected and recommended with the consideration of time efficiency. For each problem categories, these historical experiences and performances for three-layer configurations can be either automatically updated after each application, or modified manually. In light of historical experiences as well as the mapping degree, an optimal algorithm portfolio will be prepared so that users may have options either to accept the recommendation or configure the algorithms themselves, thus greatly simplifying the configuration and enhancing the quality of optimization.

Overview, the concept of DC-IOA is quite similar with hyperheuristic, which is a heuristic search method that seeks to automatically incorporate several simpler heuristics (or components of such heuristics) to efficiently solve computational search problems. But DC-IOA is different with hyperheuristic not only in combination form but also in combination characteristics. The three-level DC-IOA combines different operators in iteration-based form with both online and offline. Moreover, DC-IOA has its own characteristics which make it much flexible in the area of intelligent optimization algorithm:

- With parameter-based configuration, parameters in combined operators can dynamically changed to adapt the searching process;
- With operator-based configuration, operators can not only be combined and hybrid through the whole searching process, but also be alternatively used in only parts of the iterations, which means space combination and time combination.
- With algorithm-based configuration, algorithm can also be alternatively used through the iterations and realize algorithm-based time combination.

3.2 Case Study

The above elaborations well shed light on the detailed implementation process of DC-IOA. Based on that, a configuration platform of intelligent optimization algorithm has been established in our research. The main instruction steps in the platform will be explained as follows. In the instruction steps, the mapping rules are run out of scope of our preliminarily work temporarily because the configurable platform is just under constructing.

3.2.1 Configuration System for DC-IOA

On the basis of MFC architecture, three basic steps as shown in Figs. 3.3, 3.4, and 3.5, are given from the engineering and application perspective, in an effort to show the specific instruction of DC-IOA.

Problem specification: Some standard optimal problems are nested in the classification tree as shown in Fig. 3.3. With the radio buttons, users can choose the corresponding problem module and set their properties such as variable number, range domain and objective functions, and then choose the coding way which is internally-installed. For some large-scale complex problems, we can also derive the fitness functions according to the customized input of the objectives and restrictions and get coding along with the variable domain.

Three level configurations: As shown in Fig. 3.3, the internally-installed operators are built in classification tree just as in problem configuration view. Just a few operators can generate a huge algorithm library because any of them can combined together arbitrarily in any order. Each operator corresponds to a parameter setting view. For a single classical algorithm, only the relevant operators are needed to be selected in order with uniformly iterations. For improved or hybrid algorithm, the corresponding operators classified in different categories should be checked as needed. Through the column of order and generation setting, operator-based and algorithm-based configuration can be easily accomplished. Parameter-based configuration is carried out for each operation after the selection of operators. For higher level configuration, Customizations are constructed in the right column.

During the configurations, if new operators or new algorithms are needed, we could input it through two kinds of load methods, i.e., dynamic link library (dll) load and source load with external compilation. Only by the steps of exporting the template, programming the new algorithm and loading the code sources can help users to import a new algorithm and compare with other existing algorithms. The new algorithm can also be configured with other algorithms through the assignment of its order and generations.

Optimization execution: Fig. 3.5 shows the simulation view of the platform. In this step, the comprehensive settings, i.e. population number and evolution strategy, should be done. Then the optimal iteration process can be switched on.

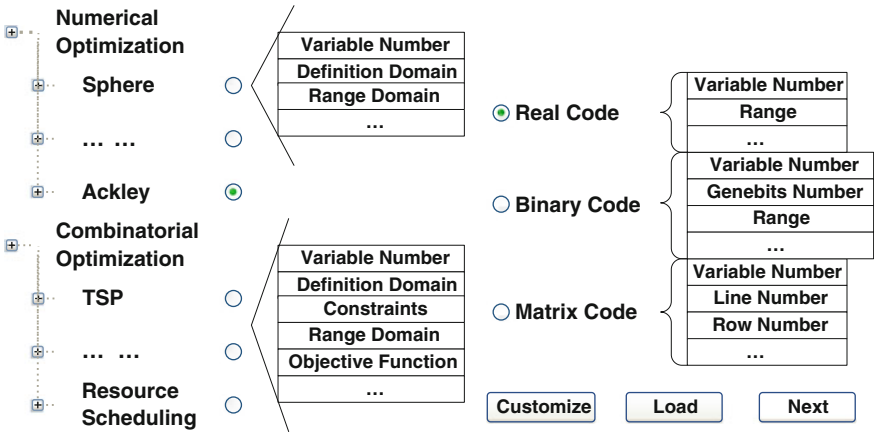


Fig. 3.3 Problem configuration view of the configurable platform

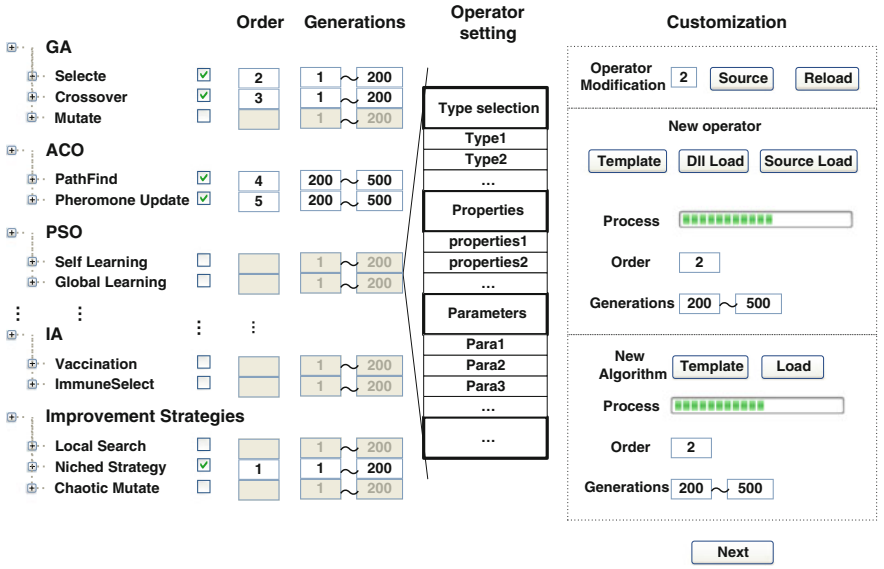


Fig. 3.4 Algorithm configuration view of the configurable platform

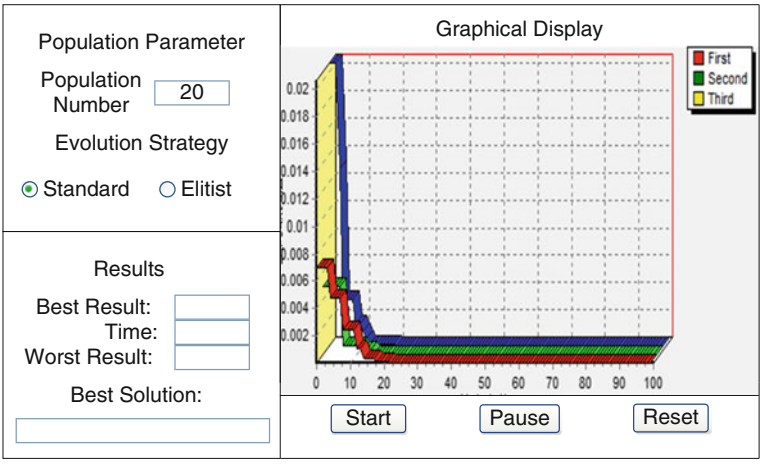


Fig. 3.5 Simulation view of the configurable platform

During the process, we can see the graphical display and the numerical results such as best solution and time consumption. And with status records in process, we can suspend it when some irregularity happens and reset the unexecuted configuration process.

According to the above three steps, most complex optimal problems can be solved within a few minutes with dynamical portfolio of various loosely-coupled operators.

3.2.2 Case Study of DC-IOA

Further, to verify the three-level configurations in the system, some typical benchmark problems are adopted in this paper for experiments. For numerical optimization, the 5 selected static benchmark functions shown in Table 3.1 and the dynamic rotation peak benchmark generator with 6 kinds of dynamic changes in [2, 3] are adopted.

In dynamic rotation peak benchmarks, we set the change frequency to be 1,000, the number of changes to be 10, the peak number to be 10. Other parameters except the above ones are the same with [3]. Moreover, in the experiments of static benchmarks, the population size is set as 50, the max generation is set as 500, while in the experiments of dynamic benchmarks, the population size and the max generation is set as 50 and 10,000 respectively. For numerical problem, the real coding method is adopted in all experiments.

Moreover, the objectives in static benchmark functions are finding the minimum optima, while the objectives (fitness function) are to minimize the offline error of dynamic rotation peak benchmarks.

For combinatorial optimization, we adopt the typical independent tasks scheduling problem which can be simplified as:

Let $D = \{d_1, d_2, \cdots, d_n\}$ be a set of design tasks, and n be the number of tasks. Let $U = \{u_1, u_2, \cdots, u_m\}$ be a set of design unit services, and m be the number of services. Then let $P(i)$ be the predecessor task set of task d_i, and $S(i)$ be its successor task set. Tasks are non-preemptive and each task can only be started after all its predecessor tasks are finished. Considering the high-priority tasks in

Table 3.1 The 5 selected test functions

Function name	Formulation
Sphere	$f_1(X) = \sum_{i=1}^{n} x_i^2, \lvert x_i\rvert \le 100$
Rastrigin	$f_2(X) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right], \lvert x_i\rvert \le 5.12$
Griewank	$f_3(X) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \lvert x_i\rvert \le 600$
Schwefel	$f_{5\_1}(X) = \sum_{i=1}^{n} x_i \sin\sqrt{\lvert x_i\rvert} + 418.9829 * n, \lvert x_i\rvert \le 500$
Rosenbrock	$f_6(X) = \sum_{i=1}^{n-1} \left[100 * \left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2\right], \lvert x_i\rvert \le 2.048$

the same design units, if u_j was selected for d_i, let $O_j(i)$ be these high-priority task set. Then

$$T_{start}(d_i) = \max_{x \in O_j(i)} (T_{end}(x)) \tag{3.4}$$

To simplify this problem, it is assumed that the "units—tasks" execution time vector as follow:

$$T_{execution(m \times n)} = \{t_{ij} | 1 \leq i \leq m, 1 \leq j \leq n\} \tag{3.5}$$

Then the objective function can be represented as:

$$f = \min T_{execution}(D) = \min T_{end}(d_n) \tag{3.6}$$

In the scheduling problem, serial connected tasks set with 30 tasks and 200 resources are selected. For getting max function, here we set the fitness function for independent scheduling to be 100/*f*. The execution time vector is generated in random way and stored in ".txt" files for uniformly testing. The reason for choosing this kind of scheduling problem is that, it is directly equivalent to resource service composition optimal selection problems in some specific environments [4–8], and both of them exist in various kinds of areas and are widely researched.

The cases we chosen are internally-installed in the platform and need not load new module. Owing to limited space, standard GA, standard ACO and 6 improvement strategies (i.e. self-adaptation operator [9] represented by A, niched operator [10] represented by N, local search operator [11] represented by L, energy-adaptation operator [12] represented by E, potential detection operator [12] represented by PD, chaotic operator [13] represented by C in algorithms) are chosen for operator-based and algorithm-based combinations. Then three experiments are carried for three-layer configurations accordingly:

(a) Based on the 5 static numerical functions, standard PSO, the improved PSO with inertia weight (WPSO) and the improved PSO with constriction factor (CPSO) are selected for parameter-based configuration. The population and iteration number are uniformly set to be 50 and 100, respectively.
Based on the 5 static numerical functions, 9 operator-based configuration portfolios and 6 algorithm-based configuration portfolios are chosen for operator and algorithm-based configuration. In algorithm-based configuration portfolios, generations are divided into 3 segments for different algorithms. On the basis of the 6 dynamic rotation peak benchmarks, 12 operator portfolios are chosen for dynamic operator-based configuration.

(b) Based on independent tasks scheduling problem, 25 operator-based configuration portfolios and 10 algorithm-based configuration portfolios with fixed order are tested. In algorithm-based configuration portfolios, generations are

divided into 2 segments which are different with experiments in numerical optimizations.

A total of 100 runs of each experiment are conducted and the average results of the three experiments are shown in Tables 3.2, 3.3, 3.4, 3.5, and 3.6 respectively.

3.2.3 Performance Analysis

In Table 3.2 it can be seen that different parameter values in the same algorithm have different performances on different problems. Within a few minutes and simple steps, we could figure out the most suitable values of c_1 and c_2 for different problem in the three algorithms, such as $(c_1, c_2) = (2.2, 2.2)$ for *Sphere* and $(c_1, c_2) = (1.8, 1.8)$ for *Rastrigin* in PSO; $(c_1, c_2, w) = (1.5, 2.5, 0.7)$ for *Rastrigin* and *Griewank* and $(c_1, c_2, w) = (2.1, 2.1, 0.7)$ for *Schwefel* and *Rosenbrock* in WPSO. Different parameters in different algorithms or operators can be set simultaneously. Users without any experiences can also find the most suitable

Table 3.2 The results of parameter-based configuration test

PSO—Parameter (c1, c2)—Value (Fitness value, Time)			
Parameter	(2, 2)	(2.2, 2.2)	(1.8, 1.8)
Sphere	**(3.74e^{-10}, 16.63 ms)**	(6.243e^{-3}, 17.26 ms)	(3.200e^{-3}, 7.763 ms)
Rastrigin	(0.7153, 5.589 ms)	(0.9644, 12.11 ms)	**(0.5621, 7.233 ms)**
Griewank	(2.114e^{-3}, 6.871 ms)	(3.757e^{-3}, 13.82 ms)	**(1.138e^{-3}, 7.903 ms)**
Schaffer	(−0.9945, 15.85 ms)	(−0.9943, 14.83 ms)	**(−0.9902, 9.191 ms)**
Rosenbrock	**(4.575e^{-2}, 6.826 ms)**	(0.12502, 10.97 ms)	(1.331e^{-2}, 15.42 ms)
The improved PSO with inertia Weight (WPSO)—Parameter (c1, c2, w)—Value (Fitness value, Time)			
Parameter	(2.1, 2.1, 0.7)	(2.1, 2.1, 0.85)	(1.5, 2.5, 0.7)
Sphere	**(6.286e^{-10}, 3.972 ms)**	(3.092e^{-5}, 9.930 ms)	(1.829e^{-9}, 4.091 ms)
Rastrigin	(0.4974, 8.110 ms)	(0.6454, 13.00 ms)	**(0.4974, 5.486 ms)**
Griewank	(2.213e^{-10}, 12.21 ms)	(5.944e^{-5}, 13.81 ms)	**(1.987e^{-10}, 6.987 ms)**
Schwefel	**(−0.9935, 8.918 ms)**	(−0.9902, 13.68 ms)	(−0.9999, 13.14 ms)
Rosenbrock	**(1.689e^{-4}, 6.247 ms)**	(1.698e^{-2}, 11.81 ms)	(2.574e^{-4}, 8.788 ms)
The improved PSO with constriction factor (CPSO)—Parameter (c1, c2)—Value (Fitness value, Time)			
Parameter	(2, 2, 50, 100)	(2.2, 2.2, 50, 100)	(1.8, 1.8)
Sphere	(3.277e^{-3}, 8.899 ms)	**(1.256e^{-26}, 10.07 ms)**	(3.874e^{-2}, 20.32 ms)
Rastrigin	**(0.7019, 5.672 ms)**	(0.4974, 10.81 ms)	(1.594, 24.13 ms)
Griewank	**(3.703e^{-3}, 6.253 ms)**	(0, 15.69 ms)	(7.479e^{-3}, 24.34 ms)
Schwefel	(−0.9902, 7.798 ms)	**(−1, 15.17 ms)**	(−0.9822, 26.29 ms)
Rosenbrock	**(5.414e^{-2}, 5.836 ms)**	(1.641e^{-10}, 13.48 ms)	(0.2321, 22.18 ms)

parameter values for the specific problems. Just with parameter-based configuration, an algorithm can be widely used in many problems without duplication and tedious modification in source code. Both algorithm beginner and employer with little professional knowledge can understand the effect of these parameters. Fine tuning of parameters becomes easier.

From the operator-based and algorithm based viewpoint, experiments in static numerical problems are taken and the results are shown in Table 3.3. The results show that the more operators we chosen, the longer the optimal process takes. On account of the different emphasis of operators, the algorithms show different performances on the same problem. If two operators which both focus on exploitation are selected, such like the portfolio of "LACO", the algorithm might easily trap into premature convergence, vice versa. Besides, it can also be seen that portfolios with more than two improvement strategies can always find better solutions, such as "NGA-AGA-L" and "C-PDE-GA", etc. On the whole, in 15 portfolios, 6 of them, i.e., LGA, PDE, PGA, PDE-LGA, NGA-AGA-L and C-PDE-GA, perform good capability in solving the 5 static benchmarks while only 3 of them, i.e., NACO, LACO and CACO, perform bad capability. From these we can see that the portfolios with operator P, DE, L and GA are quite suitable for typical static numerical functions while ACO is not suitable. Additionally, the results of 12 portfolios for dynamic rotation peak benchmarks are shown in Table 3.4. Because of the dynamic environments, heuristics are hard to define, thus ACO with heuristics is not used at all in dynamic optimization. With GA and 6 improved strategies, the whole performances of these portfolios are not better than the specific designed algorithms such as [14, 15]. But it can be seen that, with only a few old typical operators, we can get more than 12 different types of algorithms. And in the 12 tested algorithms, there are 6 have good performances relatively. We still could see from the results that adaptation strategies and local searching strategies are quite suitable for dynamic optimization while chaotic and niche strategies with high diversity are unsuitable.

Thus the selection of operators is the most important step in the configurable platform. It can not only make beginners to understand the operator clearly and help the employer to form an efficient improved intelligent optimization algorithm quickly, but also assist researchers do more comparisons for different problems and extend the application range of operators.

Likewise, Tables 3.5 and 3.6 show the operator-based and algorithm-based configuration in solving typical independent scheduling problem. The searching results of simple GA and ACO are worse than any of these compositions. In 25 kinds of operator-based portfolios, there are 7 have good performances. Although portfolios like AGA, NGA and EGA got lower fitness values, their running times are quite considerable. In 10 kinds of algorithm-based portfolios, 3 of them, i.e., ACO-GA, GA-LGA, CGA-ACO, perform good efficiency and are much better than traditional GA and ACO. That is because the former algorithms mainly focus on exploration at the previous stage, and the latter one primarily strengthen after that. Then the exploration and exploitation can be easily balanced by regulating the execution iterations of the former and latter algorithms. This new idea of

Table 3.3 The results of operator-based configuration and algorithm based configuration tests for static numerical problem

	NGA				AGA				**LGA**			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Sphere	0.5083	0	0.04	0.0136	0.0205	0.1	0.9	0.148	0.0731	0	0	0
Griewank	1.5527	0	7.9594	3.2601	0.0837	0	2.5	2.6526	0.2649	0	2.5	2.5637
Rastrigin	1.0273	0	7.9594	3.1323	0.0476	0.1	2.5	2.648	0.1595	0	2.5	2.5
Rosenbrock	0.5442	1.8223	7.9784	6.3617	0.0375	6.694	4.7047	3.9266	0.1115	3.728	7.9893	6.2219
Schwefel	0.469	0.8082	8.2361	6.0155	0.0443	2.9601	6.7053	4.4886	0.2969	00807	1.1376	0.3248
	CGA				NACO				LACO			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Sphere	0.0643	0	0.01	0.0034	3.4668	10	51.94	35.5722	3.1004	10	44.98	28.8042
Griewank	0.2501	0	2.5	2.5693	4.8197	0.756	17.9	15.0289	9.4051	0.692	13.265	11.9802
Rastrigin	0.1478	0	2.5	2.5034	6.1051	3.91	17.9	14.9337	6.3338	5.3704	13.265	12.8891
Rosenbrock	0.1171	6.205	7.9892	7.1988	3.44892	8.85	19.485	12.1962	3.0776	6.5896	18.7912	9.6381
Schwefel	0.1302	0.3398	1.7644	0.7349	4.9171	6.756	24.177	16.2193	5.4569	5.692	22.4755	14.2978
	CACO				**PDE**				**PGA**			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Sphere	3.5267	3.95	22.09	13.7604	0.0451	0	0	0	0.0535	0	0.02	0.0072
Griewank	10.1015	6.3784	12.1693	8.1373	0.1638	0	2.5	2.511	0.1823	0	2.5	2.6439
Rastrigin	6.7756	6.95	17.1693	8.4398	0.0985	0	2.5	2.5	0.1174	0	2.5	2.5072
Rosenbrock	3.0281	4.5625	15.4144	10.6393	0.0477	4.6709	7.0178	5.7485	0.0738	3.1713	7.5757	6.6425
Schwefel	5.3198	6.3784	20.6375	15.9908	0.0575	0.6085	1.6163	0.9366	0.8112	1.1128	3.854	2.4201

(continued)

Table 3.3 (continued)

	NGA				AGA				**LGA**			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
	PDE-LGA				NGA-ACO				CGA-ACO			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Sphere	0.0485	0	0	0	1.1703	0	0.08	0.0322	0.9098	0	0.06	0.014
Griewank	0.1979	0	2.5	2.5444	3.4863	0	11.9793	5.5490	2.7844	0	6.1396	3.1867
Rastrigin	0.1187	0	2.5	2.5	2.3185	0	11.9793	5.3531	1.8469	0	6.1396	3.0599
Rosenbrock	0.0904	1.9115	7.7228	6.0477	1.3785	0.9066	9.0451	6.3198	1.1443	0.5488	8.8711	6.1825
Schwefel	0.1134	0.9945	2.7276	1.6574	3.4127	1.011	12.808	4.0313	3.7448	0.6084	8.2335	4.5681
	NGA-AGA-L				NGA-EGA-ACO				**C-PDE-GA**			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Sphere	0.1922	0	0.07	0.007	1.0453	0	0.04	0.0122	0.03632	0	0.03	0.0206
Griewank	0.6204	0	2.5	2.5304	3.1409	0	4.3198	2.7739	0.13912	0	2.5	2.63915
Rastrigin	0.3987	0	2.5	2.507	2.1011	0	4.3198	2.6577	0.08364	0	2.5	2.5206
Rosenbrock	0.2111	1.5727	8.6391	6.0578	1.21	0.7961	8.7712	6.1468	0.0607	2.0754	4.6124	2.8195
Schwefel	0.234	0.0014	7.9492	4.1817	6.2074	2.8788	9.8728	7.5539	0.0724	0.5804	6.5147	2.2687

Table 3.4 The results of operator-based configuration and algorithm based configuration tests for dynamic numerical problem

	LGA				**AGA**				CGA			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Smallstep	3.1619	3.3686	16.8923	6.1997	3.124	3.5501	16.1905	6.9482	3.2037	3.8366	13.0731	7.8122
Largestep	3.1554	3.1927	15.64	6.8574	3.1115	2.4972	14.4157	5.406	3.2543	1.99	14.9302	7.3002
Random	3.1579	4.7553	17.553	7.0439	3.1163	2.8625	16.9627	6.5371	3.2469	5.1638	13.1534	6.9553
Chaotic	3.1784	4.9507	18.884	8.9015	3.1269	3.915	16.4942	7.2629	3.0905	6.8364	16.5463	8.4445
Recurrent	3.1524	3.5981	18.4097	6.5291	3.1003	4.3888	14.8048	7.8427	3.1552	5.5967	15.3695	8.3223
Recu_nois	3.1458	5.72	17.2737	8.9215	3.0643	4.6262	14.0845	6.2297	3.1281	4.2186	16.3201	8.1333
	LGA				**AGA**				CGA			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Smallstep	3.1619	3.3686	16.8923	6.1997	3.124	3.5501	16.1905	6.9482	3.2037	3.8366	13.0731	7.8122
Largestep	3.1554	3.1927	15.64	6.8574	3.1115	2.4972	14.4157	5.406	3.2543	1.99	14.9302	7.3002
Random	3.1579	4.7553	17.553	7.0439	3.1163	2.8625	16.9627	6.5371	3.2469	5.1638	13.1534	6.9553
Chaotic	3.1784	4.9507	18.884	8.9015	3.1269	3.915	16.4942	7.2629	3.0905	6.8364	16.5463	8.4445
Recurrent	3.1524	3.5981	18.4097	6.5291	3.1003	4.3888	14.8048	7.8427	3.1552	5.5967	15.3695	8.3223
Recu_nois	3.1458	5.72	17.2737	8.9215	3.0643	4.6262	14.0845	6.2297	3.1281	4.2186	16.3201	8.1333
	NGA				PGA				**PDE**			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Smallstep	3.5564	4.8375	17.7021	8.4201	1.1588	2.4155	14.2706	6.5004	1.3504	0.684	12.0536	4.4878
Largestep	3.5525	2.8943	17.0693	7.3031	1.1543	2.8687	18.6282	7.9014	1.338	0.6526	13.4898	3.4069
Random	3.5528	4.7877	15.8096	6.4017	1.1526	3.9096	16.4563	6.2729	1.3639	0.8918	12.4389	4.4398
Chaotic	5.4571	6.3704	18.6908	8.8928	1.1599	3.1	16.4825	6.5617	1.3858	1.4274	14.592	5.4005
Recurrent	5.437	5.1012	18.1462	8.1062	1.1579	3.5412	15.3805	5.9987	1.3717	1.1243	13.7095	4.9737
Recu_nois	3.5940	6.2543	18.7038	8.0757	1.1563	4.0618	16.22	6.003	1.3675	1.4499	14.8571	4.8229

(continued)

Table 3.4 (continued)

	CAGA				CPGA				**LPGA**			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Smallstep	3.0714	2.4346	16.6309	5.846	3.0578	3.679	17.2646	7.3207	3.0323	3.1235	14.3532	6.7666
Largestep	3.073	3.0775	16.1488	6.2196	3.0096	2.68	16.6254	7.5862	3.0866	2.9848	15.7231	7.1076
Random	3.0911	4.2291	15.0685	6.3011	3.0106	5.1339	14.3554	8.5794	3.1036	3.653	15.0517	7.2313
Chaotic	3.1269	5.6986	16.4942	7.2629	3.0044	6.1003	16.7793	8.5702	3.0846	4.5932	16.1378	8.4798
Recurrent	3.1003	6.3737	15.8048	6.8427	3.0048	5.4494	16.3661	8.9117	3.0777	5.3927	15.9993	7.9021
Recu_nois	5.3286	6.9901	16.5906	7.7282	3.0027	4.782	18.1317	7.1973	3.1086	5.4569	16.2109	8.5755
	CPDE				**LPDE**				**NPDE**			
	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit	Time	Bestfit	Worstfit	Avefit
Smallstep	3.264	2.7687	13.0111	6.3754	3.2628	3.0212	15.6251	6.8353	3.4329	2.8942	14.9962	6.2612
Largestep	3.242	2.4486	15.4944	7.0171	3.2136	2.1264	15.0578	6.5455	3.4787	2.5499	15.509	7.3983
Random	3.2603	4.5792	15.6206	6.8471	3.2614	4.3234	14.6317	7.9172	3.4423	4.3503	14.0945	6.4681
Chaotic	3.3060	4.1789	16.1929	7.1714	3.2889	4.915	15.6587	6.4518	3.4386	4.27	15.9813	6.677
Recurrent	3.2166	3.9007	16.8225	7.3947	3.3285	3.698	15.3388	6.9287	3.5038	3.8905	17.2054	7.3402
Recu_nois	3.2222	4.2643	15.8088	6.6107	3.2984	3.9833	16.4314	6.8012	3.4423	4.0026	16.1766	7.0424

Table 3.5 The result of operator-based configuration tests for independent scheduling problem

Operators portfolio	AGA	NGA	LGA	EGA	CGA	NAGA	**LAGA**	EAGA	**CAGA**
Fitness	11.6477	11.7155	12.0694	12.1928	11.99	11.8819	**13.042**	11.4646	**13.0809**
Time/ms	0.375	0.703	1.531	0.594	1.016	0.532	**1.484**	0.625	**0.984**
Operators portfolio	LNGA	NEGA	CNGA	LEGA	LCGA	CEGA	**NLAGA**	NEAGA	**NCAGA**
Fitness	13.0741	11.663	12.0324	11.16	12.498	12.2004	**13.5841**	12.7669	**13.2695**
Time/ms	2.203	0.672	1.157	1.985	2.235	1.187	**1.703**	0.625	**1.125**
Operators portfolio	**LEAGA**	CEGA	LCAGA	CEAGA	LNEAGA	**LCNAGA**	**NCEAGA**		
Fitness	**13.2556**	12.2004	13.4821	12.3665	13.0649	**14.0392**	**13.5517**		
Time/ms	**1.328**	1.187	2.156	1.14	1.719	**2.344**	**1.328**		

Table 3.6 The results of algorithm-based configuration tests for independent scheduling problem

Algorithms portfolio	GA	ACO	GA-ACO	ACO-GA	GA-AGA	AGA-GA
Fitness	10.9768	9.703	11.6683	**12.1472**	11.4031	11.08
Time/ms	0.36	0.8297	0.659	**0.7125**	0.375	0.438
Algorithms portfolio	GA-LGA	LGA-GA	ACO-IGA	IGA-ACO	ACO-CGA	CGA-ACO
Fitness	**13.4237**	12.2913	12.0976	11.9006	12.1452	**13.882**
Time/ms	**1.3454**	1.2397	1.7981	1.5219	1.8043	**1.6283**

combination of two or more algorithms breaks the traditional one algorithm evolution theories to fully extend the potentials of algorithm. No matter for employers or researchers, it is a very efficient way to improve the searching capability of intelligent optimization algorithms quickly and simply.

3.2.4 Comparison with Traditional Optimal Process

According to the above experiments, the advantage of our configurable platform to deal with optimization problems over traditional ways on shortening developing cycle is manifest, as shown in Fig. 3.6.

Under both two circumstances, efforts must be put on how to abstract a practical problem into mathematical formulations and what encoding manners would be matched, which normally takes a few hours. However, sequent processes may

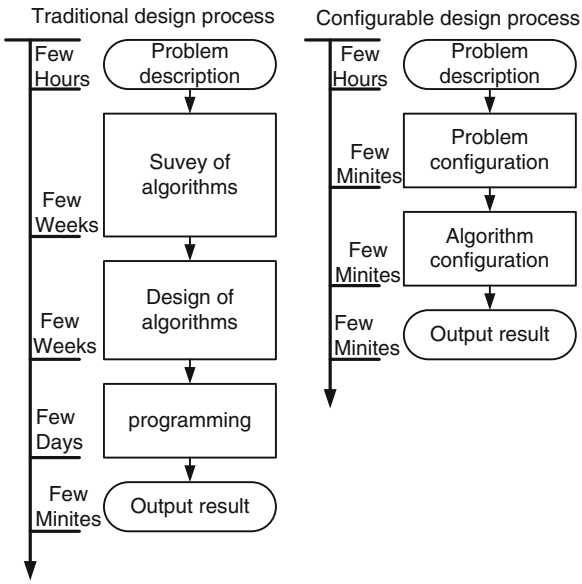


Fig. 3.6 developing cycle of traditional and configurable design of intelligent optimization algorithms

develop differently. Much energy will be invested on algorithm investigating, designing, and programming in early times, each might last for a few weeks or at least a couple of days, and ultimately add up to several months.

Yet on average, it requires much less if adopting the configuration ways with less operators. From the three experiments above we could see that dozens of improved algorithms under three-layer configurations can be generated by only 6 improved operators and 2 basic algorithms in minutes. Better solutions can not only get from parameter setting and operators selecting, but also come out from various algorithm portfolios through dividing generation process into multiple segments. Fresh algorithms will be configured dynamically by deploying different operators in the library whether you are a domain-expert or not. Keep changing the portfolios and their parameters, there is always one package right for your problem. Thus problems are expected to be resolved within at most few hours. Even if no appropriate modules are available currently, new operators or algorithms could be created via simple modifications on built-in templates or separate codes imported from outside. After that, new modules can be stored and our system will be expanded and updated, making the whole configurations and optimization process easy-adapting and time-conserving.

3.3 Summary

Improvements in metaheuristics emerge in endlessly. How to fully use them and verify their efficiency for all kinds of optimal problems are very critical. This paper summarized the current research situation and development of metaheuristics and then presented the concept of DC-IOA. To sum up, the primary works and contribution of this paper can be concluded as follows.

(1) Three-level dynamic configuration for intelligent optimization algorithms was presented. Existing operators with uniformly I/O can be arbitrarily combined to produce different types of algorithms. More importantly, the separation of generation process into several segments and employing different algorithms in different segments is a new way to produce and design intelligent optimization algorithms.
(2) Based on the new configuration ways, the mainframe and construction of DC-IOA was elaborated. And a basic configuration platform for intelligent optimization algorithms was established. The instruction steps were elaborated for further understanding. Experiments based on the platform were taken and verified the high flexibility and efficiency of it.

However, DC-IOA also has its limitations. Firstly, based on typical intelligent optimization algorithms, only parts of them are divided into operators and integrated in our DC-IOA platform. Configurations of intelligent optimization algorithms are limit. Secondly, DC-IOA has been successfully applied in static and dynamic numerical optimizations. But with less operators integrated, the

application of DC-IOA in multi-objectives are limit. The effectiveness of DC-IOA in complex problems is to be proved. Thirdly, the application of DC-IOA has not been extended in distributed environments such as grid and cloud.

Thus, future work includes firstly the construction of mapping rules and the extension of algorithms library in the platform. Without mapping rules, users can only find the best portfolio of operators by trial-and-error method. Besides, the fine-classification of optimal problems according to their variables, constraints and objectives are required for intelligent mapping. Thus efforts should be devoted both to the deeply analysis of the general optimal problems and the introduction of mapping rules. Further, for overcoming the limitations, more valuable operators should be integrated and applied in complex real-world problems. Also, the design of DC-IOA in distributed and uncertain environments such as grid (e.g., manufacturing grid system [2, 3, 6] and cloud (e.g., cloud manufacturing system [16–20] is also one of the important future works.

References

1. Tao F, Laili YJ, Liu Y, Feng Y, Wang Q, Zhang L, Xu L (2014) Concept, principle and application of dynamic configuration for intelligent algorithms. IEEE Syst J 8(1):28–42
2. Li C, Yang S (2008) A generalized approach to construct benchmark problems for dynamic optimization. Lect Notes Comput Sci 5361:391–400
3. Li C, Yang S, Nguyen TT, Yu EL, Yao X, Jin Y, Beyer HG, Suganthan PN (2009) Benchmark generator for CEC 2009 competition on dynamic optimization. Technical report, University of Leicester and University of Birmingham
4. Tao F, Zhao DM, Hu YF, Zhou ZD (2008) Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system. IEEE Trans Industr Inf 4(4):315–327
5. Tao F, Zhao D, Hu YF, Zhou ZD (2010) Correlation-aware resource service composition and optimal-selection in manufacturing grid. Eur J Oper Res 201(1):129–143
6. Tao F, Hu YF, Zhou ZD (2008) Study on manufacturing grid & its resource service optimal-selection system. Int J Adv Manuf Technol 37(9-10):1022–1041
7. Tao F, Zhao D, Zhang L (2010) Resource service optimal-selection based on intuitionistic fuzzy set and non-functionality QoS in manufacturing grid system. Knowl Inf Syst 25(1):185–208
8. Tao F, Zhang L, Nee AYC (2010) A review of the application of grid technology in manufacturing. Int J Prod Res 49(13):4119–4155
9. Srinias M, Patnaik LM (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cybern 24(4):656–667
10. Horn J, Nafpliotis N, Goldberg DE (1994) A niched pareto genetic algorithm for multiobjective optimization. In: Proceedings of the 1st IEEE World Congress on Computational intelligence
11. Ishibuchi H, Murata T (1998) A multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Trans Syst Man Cybern Part C Appl Rev 28(3):392–403
12. Laili YJ, Zhang L, Tao F (2011) Energy adaptive immune genetic algorithm for collaborative design task scheduling in cloud manufacturing system. In: IEEE international Conference on industrial engineering and engineering management, pp 1912–1916

13. Han F, Lu QS (2008) An improved chaos optimization algorithm and its application in the economic load dispatch problem. Int J Comput Math 85(6):962–982
14. Li C, Yang S (2012) A general framework of multi-population methods with clustering in undetectable dynamic environments. IEEE Trans Evol Comput 16(4):556–557
15. Yang S, Li C (2010) A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Trans Evol Comput 14(6):959–974
16. Tao F, Zhang L, Lu K, Zhao D (2012) Study on manufacturing grid resource service optimal-selection and composition framework. Enterp Inf Syst 6(2):237–264
17. Tao F, Zhang L, Venkatesh VC, Luo YL, Cheng Y (2011) Cloud manufacturing: a computing and service-oriented manufacturing model. Proc Inst Mech Eng Part B J Eng Manuf 225(10):1969–1976
18. Tao F, Cheng Y, Zhang L, Zhao D (2012) Utility modeling, equilibrium and collaboration of resource service transaction in service-oriented manufacturing. Proc Inst Mech Eng Part B J Eng Manuf 226(6):1099–1117
19. Tao F, Guo H, Zhang L, Cheng Y (2012) Modelling of combinable relationship-based composition service network and theoretical proof of its scale-free characteristics. Enterp Inf Syst 6(4):373–404
20. Tao F, Hu YF, Zhou ZD (2009) Application and modeling of resource service trust-QoS evaluation in manufacturing grid system. Int J Prod Res 47(6):1521–1550

Chapter 4
Improvement and Hybridization of Intelligent Optimization Algorithm

Algorithm improvement and hybridization are two important branches in the development of intelligent optimization algorithm. Today there are already hundreds of improvement forms in evolutionary algorithms and neighborhood search algorithms, more than 20 improvements forms in swarm intelligent algorithms and various hybridization structures. We cannot exactly count how many repetitions in these improvement and hybridization for different problems. It's harder for researchers to test all of them in different problems with different environments and compare them one by one. However, with the idea of configuration, we can extract the operators in different intelligent optimization algorithm and their improvement and hybridization forms as independent modules, recombine them and make full use of them in different problems.

In this chapter, from the perspective of algorithm improvement and hybridization, we introduce the improvements in four aspects of intelligent optimization algorithm, i.e., initialization, encoding, operator and evolutionary strategy, and elaborate the hybridizations in three aspects, that are exploration, exploitation and adaptation, as shown in Fig. 4.1. Further, the application of dynamic configuration in algorithm improvement and hybridization are detailed and discussed based on several typical intelligent optimization algorithms commonly used in manufacturing.

4.1 Introduction

Based on genetic algorithm, particle swarm optimization and ant colony optimization and so on, a number of new variations on intelligent optimization algorithm are evolved and developed [1, 2]. Generally, we classify these variations as improvement and hybridization. Improvement includes changing, increasing or deleting part of operators based on original algorithm flow, such as the parameter

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_4

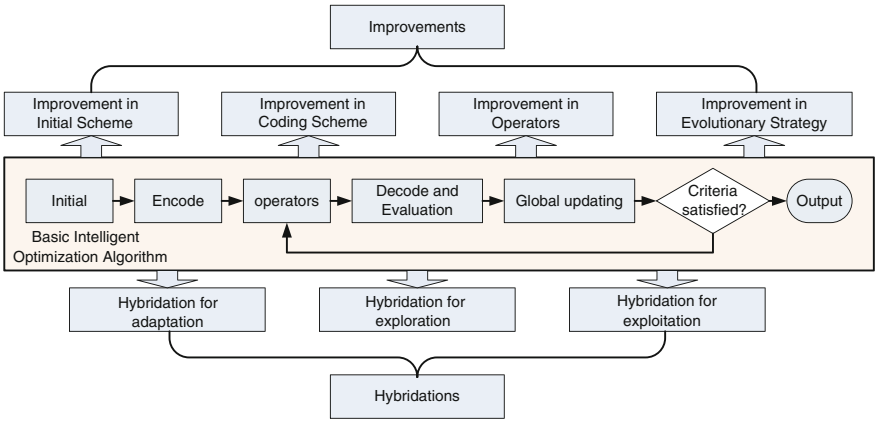


Fig. 4.1 The classification of algorithm improvement and hybridization

adaptive adjustment in crossover and mutation and the addition of niche strategy. Hybridization consists of all kinds of combinations of part of operators in different algorithms, examples are hybridization of genetic algorithm and particle swarm optimization in which the crossover and selection are applied and combined with the learning operators of particle swarm. Taken as a whole, the improvement places emphasis on the modifications in operators, while the hybridization mainly focuses on the recombination of different operators.

From the macroscopic angle, both improvement and hybridization are established for better efficiency. From the angle of specific goals, the improvement and hybridization can be further divided in 4 kinds, the reduction of time consumption, the improvement of solving accuracy, the enhancing of algorithm stability and the handling of searching convergence.

The reduction of time consumption: Many complex problems have the requirement of decision timeliness because of their changing states. Dynamic parameter adjustment in process control and live migration of tasks in production line are typical instances [3, 4]. To make sure the timeliness of the system states, people commonly choose operators with low time complexity and high exploration and simplify the complex variable relationships with fuzzy mapping [5].

The improvement of solving accuracy: As we know, algorithm cannot have the highest searching ability and the lowest time complexity both. Therefore, contrary to algorithm design for time reduction, some designers try to improve the searching ability of algorithm with the sacrifice of searching time for some problems which require higher solving accuracy. With different requirement, designers always apply some exploitation strategies and heuristics to do some local traversal search to improve algorithm searching ability [6, 7].

The enhancing of algorithm stability: For especially large-scale problems, intelligent optimization algorithm generally has some randomness. The fluctuation of solution results in several runs can be represented by algorithm stability. With

different initial population and middle random operators, the solution quality is hard to ensure. Thus the enhancing of algorithm stability becomes quite important. Researchers generally attempt to normalize the initialization and fix the searching direction to narrow the differences of results in various runs [8, 9].

The handling of algorithm convergence: With this target, some of researchers mainly focus on the avoidance of divergence which caused by operators with high exploration, low exploitation and behavior without collaboration [10, 11]. Others generally concentrate on enhancing the searching diversity to avoid the premature convergence which is responsible for low exploration and unbalanced local searching [12, 13].

According to the "no free lunch" law [14], no algorithm can obtain comprehensive performance improvement in the above four aspects simultaneously. With numerous improvements and hybridizations, the accuracy, time consumption, stability and convergence of intelligent optimization algorithms have obtained large progress in different complex manufacturing problems [15–17]. Based on the four objectives, the classification of improvement and hybridization in intelligent optimization algorithm are summarized in the following sections.

4.2 Classification of Improvement

The improvement of intelligent optimization algorithm indicates the modification of part of operators in algorithm flow. According to the uniform searching process, the improvement can fall into four sides, improvement in initial scheme, improvement in coding scheme, improvement in operator and improvement in evolutionary strategy.

4.2.1 Improvement in Initial Scheme

Initialization contains the population generation and parameter assignment. It decides the initial positions of individuals in the solution space. Good initial allocation can help the algorithm obtaining better solutions, while bad initial allocation may result in premature convergence. Besides, uneven distribution of individuals can lowering the algorithm stability to some extent, and too intensive or fixed allocation also can make the whole searching with low diversity. As described in Sect. 1.5.2, the most commonly used initial schemes include random initialization, sequential initialization and rule-based initialization.

Random initialization: It means to randomly assign values for different individuals in the domain. As the most commonly used scheme, it is independent of the specific problem and easy to implement. In this scheme, uneven allocation is one of its main drawbacks. With different implement environment and different methods of generating pseudorandom numbers, it is quite unstable. Moreover, in

large-scale solution space, a relatively small number of individuals may fall into bad allocations to a great degree. The main solution for the above situations are dividing the solution space and applying more even random scheme such as Monte Carlo or Mersenne Twister in each sub-spaces, or generating more individuals and filter better ones with good allocations [18–20].

Sequential initialization: This method refers to assign regular sequential numbers to individuals successively. During the process, sequential numbers are generated by dividing the variable domain with equal length in each dimension. It makes sure that the individuals evenly distribute in the whole solution space. Yet the fixed positions established by sequential initialization bring some drawbacks to algorithm. Firstly, it may bring about low diversity at the beginning of search. Many corners in the solution spaces are tend to become blind area. Secondly, if the distance of these individuals and the global optimal solutions are far, then better solutions can not be easily found. These may lead to premature convergence in a large degree.

Rule-based initialization: It refers to initialize the individuals according to problem property and environment-based rules. For different problems, people need to design specific rules to guide the initialization. The rules can be defined in accordance with some state forecasting and problem priori-knowledge. For instance, we can firstly divide the solution space, define some optimal positions in the sub-spaces and use these optimal positions as initial allocation. With different problem-based rules, better searching pace and solution quality can be obtained compared to the above methods. The main drawback of the rule-based initialization is that it is problem-dependent with low scalability. Some researchers also use deterministic algorithm beforehand to generate initial solutions and realize the hybridization of deterministic algorithm and intelligent optimization algorithm [21–23].

Beyond that, there are many other initialization schemes in different kinds of intelligent optimization algorithms [24, 25]. On account of the low efficiency of sequential initialization, random initialization is the most widely applied because it is more likely to be implemented. Moreover, although rule-based initialization scheme is hard to design, it is also broadly used in engineering for better searching ability. In some cases, the hybridization of random initialization and rule-based initialization might be a great way to obtain a good initial allocation with high diversity.

4.2.2 Improvement in Coding Scheme

Encoding are defined as the transformation or mapping between individual genes and problem variables. Except direct real number coding scheme, almost all other coding ways need extra time to execute transformation in iteration, which can straightforwardly increase time complexity. The major influences of encoding scheme are increasing algorithm diversity and enhancing the searching ability by

cooperating with operators. Thus efforts on encoding primarily aim at the improvement of solving accuracy and the handling of algorithm convergence. As described in Sect. 1.4.1, most typical coding schemes are designed based on genetic algorithm and gradually generalized to other intelligent optimization algorithms. At present, many newly-developing encoding schemes are established for specific manufacturing optimization problems [26, 27]. Yet the most applied coding scheme in engineering are still real coding, binary coding and matrix coding and so forth [28, 29]. Here we list some representative ones.

Real coding: As introduced above, no matter in continuous or discrete optimization, the encoding and decoding operations in iteration can be avoided. It is suitable for big variables and can effectively reduce the time complexity of algorithm.

Binary coding: In this scheme, variable in each dimension is mapped as a group of binary (or Boolean) genes. It is the eldest coding scheme in intelligent optimization algorithm. When applying it, the length of genes for each variable must be defined previously. When the accuracy requirement is higher, we need very long genes to present big variables. Hence, compared with real coding, it will largely increase the searching time especially in large-scale problems with big variables and high accuracy requirement.

Matrix coding: As the same as binary coding, each variable can be represented by a line of individual genes. The genes can be either Boolean or real number. It is designed especially for discrete combinatorial optimization. Typical time sequence, symbol string and multi-dimension position can all be directly represented by matrix coding. It is more flexible than the above coding scheme, yet will take more memory space and more complex.

Quantum coding: It is a new coding scheme which is used very frequently in recent research. Borrow the dimorphism of dual quantum genes and quantum rotation door, it maps the variables to quantum genes by two steps. This can bring high diversity to population during the whole searching process. However, dimorphic quantum genes will also bring two step operations both in encoding and decoding and take more memory space in the programming, so as to lower the decision efficiency.

With changing environment in production and manufacturing system, researchers and engineers have made great efforts on digging new encoding schemes [30, 31]. No matter in what coding scheme, the main consideration is the encoding and decoding complexity without the loss of searching accuracy. In different coding scheme, especially for combinatorial optimization, we may easily get into a case that multiple individuals mapping as one single solution, such as real coding scheme in job shop scheduling problem. In such situation, repetitive searching and uneven pace become inevitable, which further lower the whole searching quality of algorithm. Therefore, finding solutions for different kinds of many-to-one cases and designing efficient one-to-one mapping coding scheme are quite imperative in more and more complex manufacturing optimization.

4.2.3 Improvement in Operator

Operators are the core of intelligent optimization algorithm, so that improvement in operators attracts more attentions than other aspects. For exploration, random regeneration, chaotic changing, niche strategy [32, 33] and so on gives the algorithm more dynamics, but with less regularity, better solutions are hard to find and the algorithm stability will be very low. For exploitation, heuristics such as priori-knowledge, local search and greedy strategy [6, 34, 35] and so on brings more local searching ability which nevertheless makes the algorithm easy to trap into local convergence and do more repetitive searching. Only if two of them combined together can they establish a balanced searching pace and good solving efficiency. Today, many researchers focus on the balanced searching of intelligent optimization algorithm and design several exploration-oriented, exploitation-oriented and adaptive operators for solving complex problems. From the improvement ways we can divide the operator improvements into the adjustment of control parameter, the modification of operation and the increase of new independent operators.

The adjustment of control parameter: Control parameter in intelligent optimization algorithm, such as crossover and mutation probability in genetic algorithm, decides the strength of operations in iteration. Therefore, for dynamic problems, people are likely to design adaptive parameter control strategy in operators in line with population searching state to guide the operations. Typical examples are the weighted particle swarm optimization, the adaptive genetic algorithm and so forth. In the adaptive genetic algorithm, the crossover and mutate probabilities are decided by the average fitness value and the best fitness value of the whole population. If the average fitness value is close to the best fitness value, the two probabilities will become lower, so as to do more exploitation. If the average fitness value is far from the best fitness value, or the differences between the individuals are large, then the two probabilities will be higher in order to strengthen the crossover and mutation and generate more new diverse individuals. It can be clearly seen that changing weights and parameters can not only balance the exploration and exploitation during search, but also adapt the algorithm for different problem environments. Except the above mentioned parameter-based improvement, fuzzy adaptive theory and knowledge-based adaptive theory can both be introduced to different sorts of operators to enhance the adaptive ability of algorithms for complex problems. Many experiments and applications have verified that such adjustments in iteration can successfully balance most of intelligent optimization algorithms during their searching pace without the increase of time complexity [36, 37]. Thus they are widely applied in engineering.

The modification of operation: It primarily refers to the structure modification in operators according to different problem requirements and the coding scheme in earlier stage. The most representative ones are multi-point crossover, chaotic mutation, discrete particle swarm learning and record list-based path finding and so on. Some of them are inclined to magnify the operation strength or range to increase algorithm diversity. Some of them are apt to choose part of individuals and add new

operations to enhance the algorithm excavating ability. These modifications noted above are normally independent of specific problem, so that they can be widely used in different environments. Except that, there are also some problem oriented modifications, such as the adding of priori information in ant colony optimization and immune algorithm, which mainly aim at increase the searching accuracy and make the algorithm more suitable for a specific problem. The difference with the above general modification is that the problem-oriented modification can only applied in a particular environment for a particular decision. At the same time, many people additionally focus on modifying the existing operators to adapt the specially designed coding scheme, such as the operator improvement of discrete particle swarm optimization for the particular coding scheme of job shop scheduling and the new record list improvement in ant colony optimization for solving continuous problems. In this point, they may basically change the operational mechanism with some new customized strategies. From all these improvements it can be seen that, when the problem to be solved and its encoding scheme are certain, the operators can all be divided into several sub-modules with population inputs and outputs according to the iterative steps. With the variation of these sub-modules, the searching direction and ability can both be changed.

The increase of new independent operators: Because the existing operators in an algorithm may possess weak capability either in exploration, or in exploitation, researchers generally tend to increase new independent operator to compensate the algorithm in a certain aspect. It is similar to algorithm hybridization, but unlike hybridization, the new operators are newly designed in accordance with the existing ones and do not belong to other algorithms. Characteristic cases are niche strategy, local search strategy and sorting based strategies and so on. For instance, niche strategy are designed before individual optimal selection, which use the hamming distance to get similar individuals and multiple a penalty function to make the weak ones been weeded out. Besides, local search strategy which tries to traverse a small range of space and find the best solution can be widely applied in different algorithms in any step and increase the algorithm exploitation. Most of these augmentations will increase the time complexity of algorithm to some extent, thus they can only be used in the problems which have low requirement on decision timeliness.

In recent times, there are still more and more improved operators springing up for diverse decision scenes and being cross-utilized in a bunch of manufacturing problems [38, 39]. But most of them have only been verified in merely one or two specific problem in theory and tests, not in wider practice.

4.2.4 Improvement in Evolutionary Strategy

Evolutionary strategy are executed after operators in iteration for updating the population and record the best and worst positions during the whole searching process. Since the elite evolutionary strategy is widely applied, it is the least

studied component of intelligent optimization algorithm because its influence on algorithm is lower than the above parts. However, with filtrating of new individuals to replace old ones, unbalanced updating will largely slow down the searching speed and may lower the whole efficiency or even give rise to premature convergence. Thus it is also very important. Improvement research in evolutionary strategy is inclined to consider the enhancing of algorithm stability and the handling of algorithm convergence and handle single-objective and multi-objective problems in different manners.

For single objective optimization, evolutionary strategy is developed from the direct replace manner to the famous elite strategy. In elite strategy, if the best individual obtained by several operators are better than the global best record, then replace the global best one with the new best one, or we should replace the worst individual in current generation by the new best one. Elite strategy becomes almost a uniform basic part in classical intelligent optimization algorithm for simple-objective optimizations. Furthermore, typical improved evolutionary strategies also include the method that combining the new individuals and the old ones and doing filtration after the combination by fitness based sorting or hamming distance based selection. These strategies play an important pre-selection role to prevent population diverge and premature convergence.

For multi-objective optimizations with Pareto scheme, elite strategy becomes not that suitable. Instead, the combination of new and old individuals in generation and screening in frontier Pareto-set is used for population updating. Now the most widely applied strategy is non dominated sorting. It is similar with the fitness based sorting in single-objective optimization. The only difference is that it uses the non-dominating theory to separate the individuals into sorted layers. Additionally, for improving the population diversity, some probability-based elimination mechanisms are also used to delete similar individuals in a specific layer randomly. In this manner, if the randomly generated number is smaller than a threshold, then accept the corresponding individual goes into the next generation, or it will be eliminated. It can increase the searching diversity and ensure the algorithm convergence in actual fact.

In a word, with some certain manners, evolutionary strategy can also be modified and further enhance the algorithm searching ability. As a key assistant part in intelligent optimization algorithm, it is quite independent with specific problem and worth to be researched for improving the efficiency of general intelligent optimization algorithms.

4.3 Classification of Hybridization

Hybridization refers to recombine or rearrange parts of intelligent optimization algorithms to generate a new method. Because a large proportion of initialization and coding schemes can be widely used in intelligent optimization algorithms, the design of hybridization chiefly focus on the recombination or rearrangement

among operators in iteration. With different emphasis, hybridization can also be divided into three kinds, hybridization for exploration, hybridization for exploitation and hybridization for adaptation.

4.3.1 Hybridization for Exploration

Hybridization for exploration mostly designed to search wider solution space in a limited time. In such category, searching step and range-ability of algorithm is quite large, which keeps the population with high diversity. Characteristic operators for exploration are mutation, differential evolution operators and taboo search and so forth [40–42]. The following are brief reviews on these typical operators for hybridization.

Mutation, originate from genetic algorithm, is one of the mostly applied operators in hybridization. It can be independently used in various intelligent optimization algorithms to avoid premature convergence. For example, in particle swarm optimization, after the global and self learning operations finished, mutation can be employed to balance the high learning-oriented operators and generate some new individuals during iteration. Similarly, if we apply the mutation operator to immune clone algorithm, parts of population can break away from the clone rule and do search further. It can be said the mutation operator is a universal operator for improving exploration in intelligent optimization algorithm. Moreover, it should be noted that when using mutation in hybridization, the probability and range of it should be concerned with the environment changing. Because too larger mutation probability and range may bring about searching diverge, while too small operation may not work for improvement.

The operator of random differential evolution is also a typical one in hybridization for exploration. With differential computing of three randomly chosen individuals, it can used to enhance the exploration ability of algorithm. By controlling a differential factor, it can amplify the exploration step to enhance population diversity or shrink the step to realize mutual learning. More flexible than mutation, it can make the new individual evenly distributed in the solution space and realize more balanced exploration. For instance, if we combine the differential evolution operator with the path finding operator of ant colony optimization, after the path finding, the differential computing with three randomly individuals can effectively alleviate the premature convergence brought about by unevenly pheromone. Except that, we can also replace the mutation in genetic algorithm with differential computing to obtain an effective hybridization taking the problem environment into account.

Taboo search are also a most commonly used operator for exploration-oriented hybridization. Different with the above two independent operators, we need to additionally design new taboo list for each hybridization scheme in different problems. If we apply taboo search in genetic algorithm, we need to design the problem-specific taboo list, do taboo judgment after genetic operators (i.e. selection,

crossover and mutation) and update the taboo list with new records. The same principles can apply to other hybridizations such as taboo-based immune algorithm and taboo-based simulated annealing algorithm. Furthermore, we can not only design taboo list for individuals, but also for other searching parameters. It makes sure the diversity during the whole searching process and avoids repetitive iteration to some extent. The main drawbacks of applying taboo strategy in hybridization are, (1) taboo judgment during iteration may bring larger time complexity, (2) long taboo list may increases the space complexity. At present, it is still a good strategy for doing wider exploration in different intelligent optimization algorithms.

4.3.2 Hybridization for Exploitation

Hybridization for exploitation is designed after exploration to make the algorithm searching in local scope more efficiently. In hybridization for exploitation, most heuristics and local search operators are applied for searching in a narrow scope. Characteristic operators are immune operators, learning operators of particle swarm optimization and path finding operators of ant colony optimization [43–45].

Immune operators include two kinds, immune clone and immune vaccination. Immune clone tries to extract features of the global best individuals to guide the operation, while immune vaccination tend to use priori knowledge of the specific problem to change parts of individuals locally. Both of them need some rules to guide and are independent from other operations. Therefore, they can be used in anywhere with some exploration-oriented operators to form a new algorithm with good exploitation ability. However, because of the problem-dependent rules, not only the design of hybridization based on immune operators will become complex, but also the searching with iterative immune vaccination and extraction will take much longer time. In other words, they sacrifices time for quality. Many studies prove that with problem-based guidance, immune operators can truly bring great searching ability and realize combination of deterministic and non-deterministic decision with high accuracy.

Learning operators of particle swarm optimization include global learning and self learning. With the record of self-best and global-best positions, they can only be applied together to balance the searching pace. With single self-learning, individuals without cooperation will become chaos and cannot get evolution any more. In reverse, with single global-learning, the population will converge quickly into a bad position. Thus, only the integration of them can ensure the stable searching. Generally, they can be used with other operators as exploitation-enhanced algorithms to improve the cooperative and stable searching. For example, they can be introduced to chaotic optimization or genetic algorithm. After the original action, the learning operators can reduce the randomness and add some guidance for exploitation. Similar with immune operators, due to the record of self-best in each generation, both the time complexity and space complexity will be increased.

Path finding operator of ant colony optimization tries to generate new population according to the pheromone concentration and do pheromone updating after that. It is also similar to the above learning operators, which attempts to use self and global record to guide the search. With pheromone updating and recording, it also has high complexity in hybridization. The most famous hybridization schemes are the combination of ant colony optimization and genetic algorithm and the simulated annealing ant colony optimization, and so on. Owing to its high complexity, it generally recombined with some simple exploration-based operators in design. For example, if we combine path finding operator with crossover and mutation of genetic algorithm, it can reduce the randomness in individual interaction and improve the local exploitation with group cooperation.

At present, these exploitation-based operators are mostly recombined with crossover, mutation and simulated annealing operators. With lower exploration and higher exploitation in local scope, these hybridizations have performed good exploitation ability in different manufacturing problems. However, with high complexity in both design and execution, how to design a simplified exploitation-based operator without accuracy loss is also an essential problem.

4.3.3 Hybridization for Adaptation

Hybridization for adaptation is mainly designed for adaptively solving problems with dynamic or complex solution space. Take the consideration of the balance of exploration and exploitation, most researchers recombine more than two operators to get high exploration in earlier stage and turn to exploitation in later stage. Generally, many intelligent optimization algorithms lack the ability either in exploitation or in exploration. Thus the above two kinds of hybridizations attempt to design hybridization to fill the gaps. But those strategies may still have the unstable or unbalanced problems. For overcoming the weakness, hybridization for adaptation receive a lot of attentions.

The mostly used operators for adaptation are simulated annealing operator. It particularly refers to the annealing acceptance judgment used in various hybridizations. In early phases, the annealing acceptance probability is quite large so that many degraded individuals can be accepted for high diversity and exploration. With the decreasing temperature, the lowering probability makes lesser and lesser degraded individuals to be accepted, so as to improve the algorithm convergence on the other side. With annealing probability, it can control the whole searching process adaptively with other exploration or exploitation-oriented operators to perform balanced searching. Moreover, it is easy to implement without bring more complexity. Thus, simulated annealing based hybrid intelligent optimization algorithms are more and more designed and used for different problems as well.

Except that, more and more efforts are putting in design adaptive operators for hybridization or recombine adaptive operators for the balance of exploration and

exploitation [46–48]. The aim of hybridization for adaptation can be summarized as getting good ability and balance in exploration and exploitation with less searching time.

4.4 Improvement and Hybridization Based on DC-IA

We have listed the classification of improvement and hybridization of intelligent optimization algorithm and their typical examples in the above sections. For different kinds of manufacturing optimizations, although with the same operations, they perform totally different. People need to do a lot of work to repetitive design various improvements and hybridizations for specific problems. For reducing these repetition and make the existing algorithms more efficiently in diverse environments, we presented a new dynamic configuration strategy mentioned in the previous chapter. With the new strategy, we can separate the solving process as four modules, i.e. initialization, encoding, operators and population updating. The input and output of each module are both population. Moreover, we can also divide and stored these modules according to their features. With two kinds of classifications, improvement and hybridization based on DC-IA can be divided into two styles, (1) module-based improvement and hybridization, (2) process-based improvement and hybridization.

Module-based improvement and hybridization means to design uniform iterative operations with new or recombined modules. It comes down to operator-based configuration. In detail, it can be classified as cascade configuration, parallel configuration and mixed configuration, as shown in Figs. 4.2, 4.3 and 4.4.

Similar with traditional hybridization, cascade configuration means to simply select multiple modules with different functions and assign operational order for them. In this scheme, one single hybrid algorithm is generated without generation division and population division. Based on the method of DC-IOA, this way can largely improve the reutilization rate of operators and produce new hybridizations directly.

Different with cascade configuration, parallel configuration refers to divide population into several sub-populations and select groups of operators for different sub-populations. In such scheme, more operators can be combined to do diverse search. Specifically, with uniform initialization and encoding scheme, population is randomly divided into sub-populations. During iteration, each sub-population is modified by different group of operators. After all operators finished, the sub-populations are combined and evolved together with uniform evolutionary strategy and randomly divided again to sub-populations. Diverse groups of operators guarantee the diversity of the whole population, while the random division of sub-populations ensures each individual can be evenly updated by different groups of operators. The whole process is balanced and more flexible than cascade configuration. Moreover, more operators will not increase the whole time complexity of algorithm as a result of the multi sub-population scheme. The main drawback is

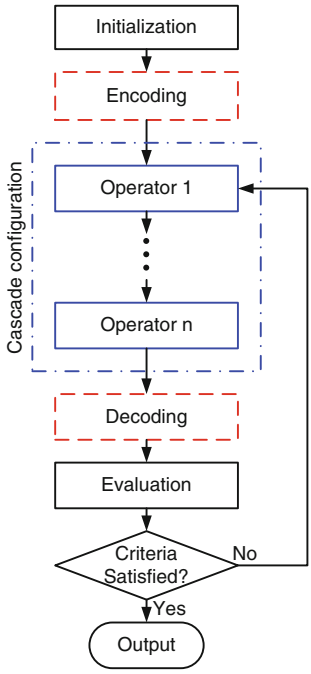


Fig. 4.2 Cascade configuration in module-based improvement and hybridization

that, it is hard to select suitable group number and design operators for several groups with high the coordination.

Further, based on cascade and parallel configuration, we can quickly implement mixed configuration. That is to say, we can perform cascade configuration with ordered simple operators and divide population into several sub-populations for parallel groups of operators. Sometimes, single cascade configuration is simple, stable but with low flexibility and diversity, while single parallel configuration ensures high diversity but inversely with low stability and interoperability. In such cases, we can combine cascade and parallel configuration, use cascade operators to enhance the interoperation and stability among sub-populations and perform parallel groups of operators to improve searching flexibility and diversity. However, it is harder to design than parallel configuration and the complexity is much higher.

Except that, we can perform dynamic configuration from the perspective of algorithm process, i.e. process-based improvement and hybridization. Different with the above module-based configuration, we can divide the generation process into several parts. Each part, containing multiple generations, performs different group of operators. It can be boiled down to algorithm-based configuration. Based on this frame, the process-based improvement and hybridization can be further divided as (1) process-based design with homogeneous coding and (2) process-based design with heterogeneous coding, as shown in Figs. 4.5 and 4.6.

In the first kind, the coding way in the whole process is unchanged. With uniform coding style, the process can be separated to two or more phrases. Each

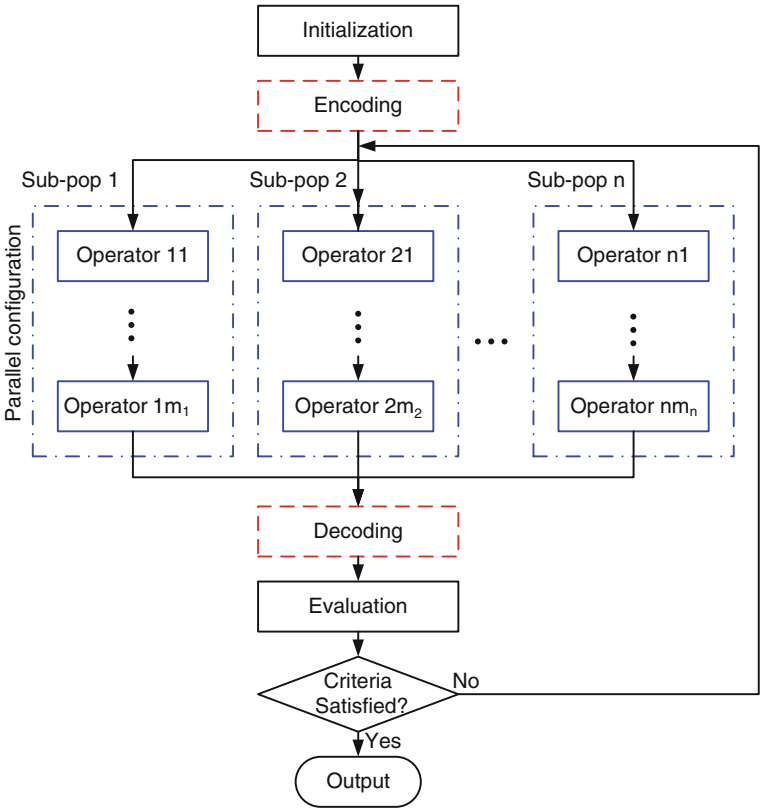


Fig. 4.3 Parallel configuration in module-based improvement and hybridization

phrase applies different group of operators. For example, with uniform binary coding scheme, we can apply genetic algorithm in generation 1–100, employ particle swarm optimization in generation 101–200 and perform ant colony optimization in generation 201–500, just as we described in Chap. 3. This configuration way is more flexible and simple than the module-based way, because users can easily obtain exploration in early stage, balance searching in middle stage and exploitation in later stage by using operators with corresponding functions, so as to make the process easier to control. In addition, with partition of searching process, the total time complexity will not be increased as well. For users with existing operators, this configuration way is much easier to design an efficient improved or hybrid algorithm than others.

Based on the process-based strategy, we can also use heterogeneous coding schemes in different parts of process. The only different is that it needs several trans-coding steps among different phrases. This allows us to applied more types of operators to further enhance diversity of operations. However, this will also largely increase the time complexity and decrease the searching efficiency.

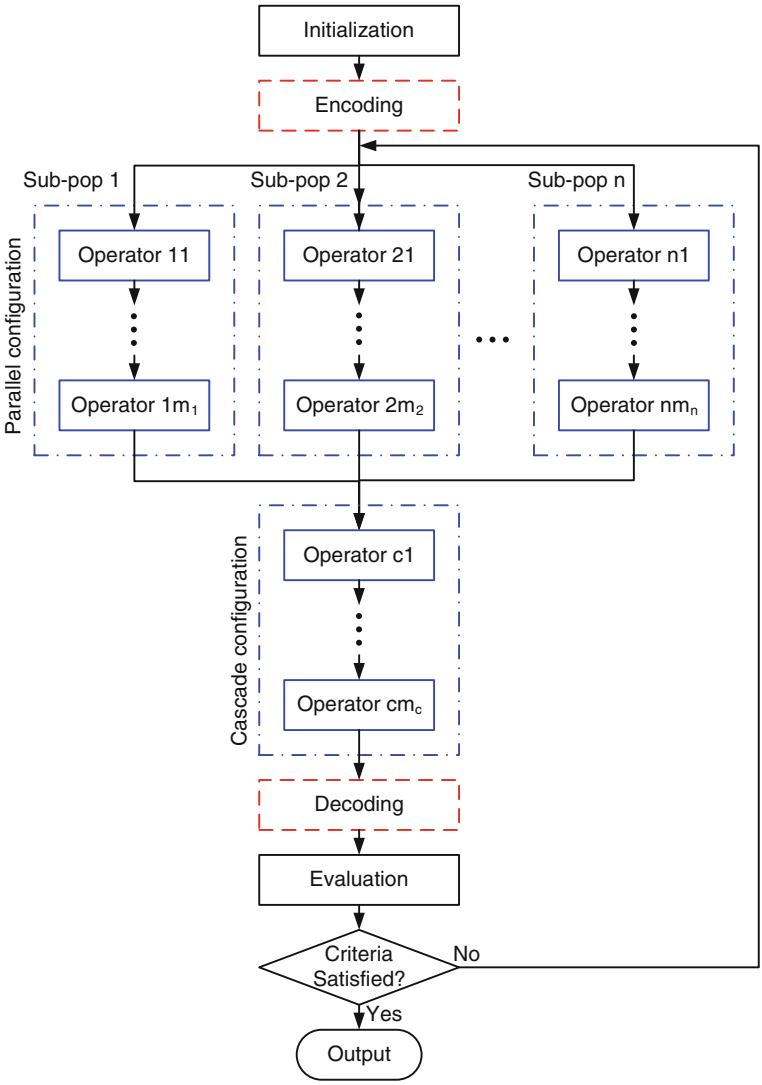


Fig. 4.4 Mixed configuration in module-based improvement and hybridization

It is worth noting that, for large-scale complex manufacturing optimization, we can combine both module-based configuration and process-based configuration to design improvement and hybridization. The design and test process in this case will become much harder. But with population and process divisions, the time complexity will not be increased much.

It can be seen, based on the concept of DC-IOA, with module-based and process-based improvement and hybridization, limited operators with uniform

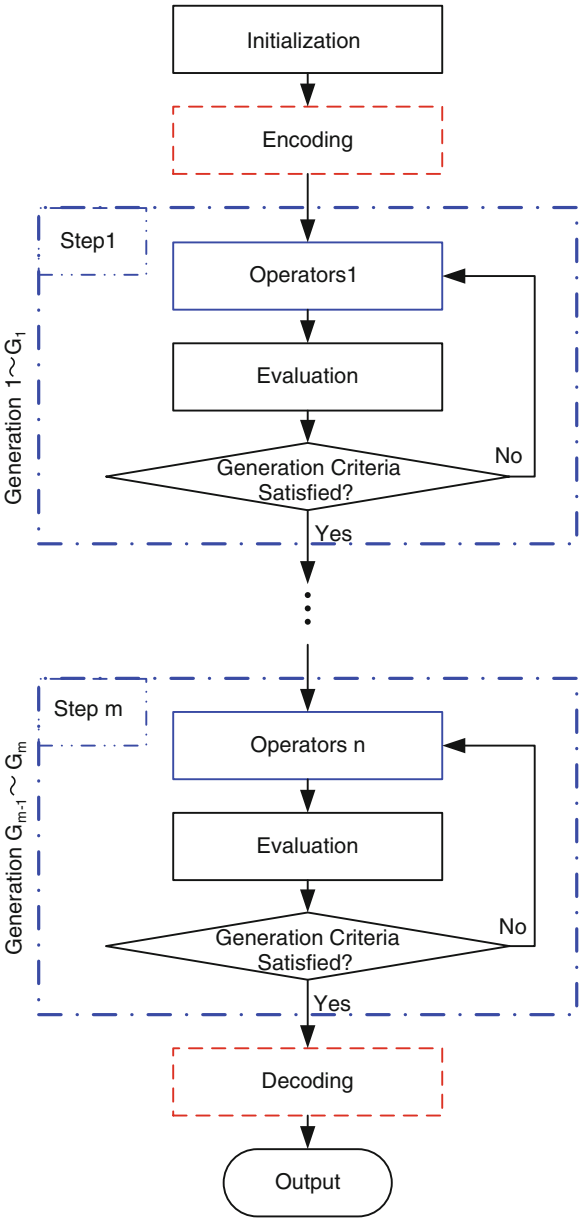


Fig. 4.5 Process-based improvement and hybridization with homogeneous coding

input and output can form hundreds of intelligent optimization algorithms with various types of structures. Not only the reutilization of existing operators is increased, but also a bunch of new dynamic improvement and hybridization schemes can be effectively applied for different kinds of problems.

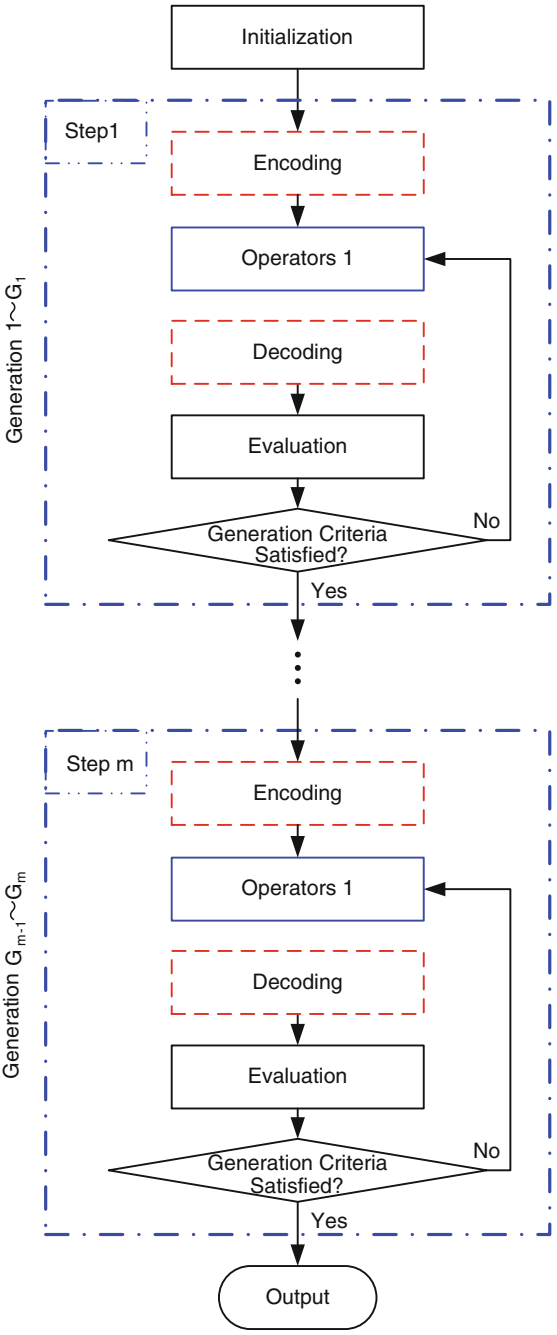


Fig. 4.6 Process-based improvement and hybridization with heterogeneous coding

4.5 Summary

Improvement and hybridization of intelligent optimization algorithm are always a focal point in the optimization of the whole life-cycle manufacturing. More than half of researches employs different sorts of improved or hybrid intelligent optimization algorithms in manufacturing optimizations, such as [49–53] and so on. This chapter introduced the improvement and hybridization of intelligent optimization algorithm, classified the improvement into four categories, i.e. improvement in initialization, improvement in coding scheme, improvement in operators and improvement in evolutionary strategy, and divided the hybridization into three kinds, i.e. hybridization for exploration, hybridization for exploitation and hybridization for adaptation. Then the detailed characteristics of the above categories are given with typical examples respectively.

Based on the existing manners and the new DC-IOA concept, we further present two kinds of new strategies for improvement and hybridization, (1) module-based improvement and hybridization and (2) process-based improvement and hybridization. The design process and features of the two kinds are elaborated. They are very practical and flexible and can play a guiding role in design and optimization in manufacturing. With the new design strategies, the flexibility and reusability of existing operators for different sorts of complex problems can be fundamentally enhanced.

References

1. Raidl GR (2006) A unified view on hybrid metaheuristics, hybrid metaheuristics. Lect Notes Comput Sci 4030:1–12
2. Parejo JA, Ruiz-Cortes A, Lozano S, Fernandez P (2012) Metaheuristic optimization frameworks: a survey and benchmarking. Soft Comput 16(3):527–561
3. Trappey AJC, Trappey CV, Wu CR (2010) Genetic algorithm dynamic performance evaluation for RFID reverse logistic management. Expert Syst Appl 37(11):7329–7335
4. Rao RV, Pawar PJ (2010) Parameter optimization of a multi-pass milling process using non-traditional optimization algorithms. Appl Soft Comput 10(2):445–456
5. Shen C, Wang L, Li Q (2007) Optimization of injection molding process parameters using combination of artificial neural network and genetic algorithm method. J Mater Process Technol 183(2–3):412–418
6. Moslehi G, Mahnam M (2011) A pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. Int J Prod Econ 129(1):14–22
7. Yildiz AR (2013) Hybrid taguchi-differential evolution algorithm for optimization of multi-pass turning operations. Appl Soft Comput 13(3):1433–1439
8. Burnwal S, Deb S (2013) Scheduling optimization of flexible manufacturing system using cuckoo search-based approach. Int J Adv Manuf Technol 64:951–959
9. Yildiz AR (2009) An effective hybrid immune-hill climbing optimization approach for solving design and manufacturing optimization in industry. J Mater Process Technol 209(6):2773–2780
10. Duran N Rodriguez, Consalter LA (2010) Collaborative particle swarm optimization with a data mining technique for manufacturing cell design. Expert Syst Appl 37(2):1563–1567

11. Wang JQ, Sun SD, Si SB, Yang HA (2009) Theory of constraints product mix optimization based on immune algorithm. Int J Prod Res 47(16):4521–4543
12. Caponio A, Cascella GL, Neri F, Salvatore N, Sumner M (2007) A fast adaptive memetic algorithm for online and offline control design of PMSM drives. IEEE Trans Sys Man Cybern B Cybern 37(1):28–41
13. Yang WA, Guo Y, Liao WH (2011) Multi-objective optimization of multi-pass face milling using particle swarm intelligence. Int J Adv Manuf Technol 56(5–8):429–443
14. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
15. Chandrasekaran M, Muralidhar M, Krishna CM, Dixit US (2010) Application of soft computing techniques in machining performance prediction and optimization: a literature review. Int J Adv Manuf Technol 46(5–8):445–464
16. Tiwari MK, Raghavendra N, Agrawal S, Goyal SK (2010) A hybrid taguchi-immune approach to optimize an integrated supply chain design problem with multiple shipping. Eur J Oper Res 201(1):95–106
17. Chan KY, Dillon TS, Kwong CK (2011) Modeling of a liquid epoxy molding process using a particle swarm optimization-based fuzzy reguression approach. IEEE Trans Industr Inf 7(1):148–158
18. Goicoechea HC, Olivieri AC (2002) Wavelength selection for multivariate calibration using a genetic algorithm: a novel initialization strategy. J Chem Inf Model 42(5):1146–1153
19. Zainuddin N, Yassin IM, Zabidi A, Hassan HA (2010) Optimizing filter parameters using particle swarm optimization. In: The 6th international colloquium on signal processing and its applications (CSPA) pp 21–23, May 1–6
20. Wang CM, Huang YF (2010) Self-adaptive harmony search algorithm for optimization. Expert Syst Appl 37(4):2826–2837
21. Zhang Y, Li X, Wang Q (2009) Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. Eur J Oper Res 196(3):869–876
22. Hong SS, Yun J, Choi B, Kong J, Han MM (2012) Improved WTA problem solving method using a parallel genetic algorithm which applied the RMI initialization method. In: The 6th international conference on soft computing and intelligent systems, vol 20–24, pp 2189–2193
23. Yao HM, Cai MD, Wang JK, Hu RK, Liang Y (2013) A novel evolutionary algorithm with improved genetic operator and crossover strategy. Appl Mech Mater 411–414:1956–1965
24. Kazimipour B, Li X, Qin AK (2013) Initialization methods for large scale global optimization. IEEE Congr Evol Comput 20–23:2750–2757
25. Dimopoulos C, Zalzala AMS (2000) Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons. IEEE Trans Evol Comput 4(2):93–113
26. Fumi A, Scarabotti L, Schiraldi MM (2013) The effect of slot-code optimization in warehouse order picking. Int J Eng Bus Manag 5(20):1–10
27. Tao F, Zhang L, Zhang ZH, Nee AYC (2010) A quantum multi-agent evolutionary algorithm for selection of partners in a virtual enterprise, CIRP Ann Manuf Technol 59(1):485–488
28. Oysu C, Bingul Z (2009) Application of heuristic and hybrid-GASA algorithms to tool-path optimization problem for minimizing airtime during machining. Eng Appl Artif Intell 22(3):389–396
29. Lv HG, Lu C (2010) An assembly sequence planning approach with a discrete particle swarm optimization algorithm. Int J Adv Manuf Technol 50(5–8):761–770
30. Kuo CC (2008) A novel coding scheme for practical economic dispatch by modified particle swarm approach. IEEE Trans Power Syst 23(4):1825–1835
31. Bhattacharya A, Kumar P (2010) Biogeography-based optimization for different economic load dispatch problems. IEEE Trans Power Syst 25(2):1064–1077
32. Laili YJ, Tao F, Zhang L, Cheng Y, Luo YL, Sarker BR (2013) A ranking chaos algorithm for dual scheduling of cloud service and computing resource in private cloud. Comput Ind 64(4):448–463

33. Perez E, Posada M, Herrera F (2012) Analysis of new niching genetic algorithms for finding multiple solutions in the job shop scheduling. J Intell Manuf 23(3):341–356
34. Prakash A, Chan FTS, Deshmukh SG (2011) FMS scheduling with knowledge based genetic algorithm approach. Expert Syst Appl 38(4):3161–3171
35. Tasgetiren MF, Pan QK, Suganthan PN, Buyukdagli Q (2013) A variable iterated greedy algorithm with differential evolution for the no-idle permutation flow shop scheduling problem. Comput Oper Res 40(7):1729–1743
36. Valente A, Carpanzano E (2011) Development of multi-level adaptive control and scheduling solutions for shop-floor automation in reconfigurable manufacturing systems. CIRP Ann Manuf Technol 60(1):449–452
37. Ye A, Li Z, Xie M (2010) Some improvements on adaptive genetic algorithms for reliability-related applications. Reliab Eng Syst Saf 95(2):120–126
38. Tao F, Qiao K, Zhang L, Li Z, Nee AYC (2012) GA-BHTR: an improved genetic algorithm for partner selection in virtual manufacturing. Int J Prod Res 50(8):2079–2100
39. Azadeh A, Miri-Nargesi SS, Goldansaz SM, Zoraghi N (2012) Design and implementation of an integrated taguchi method for continuous assessment and improvement of manufacturing systems. Int J Adv Manuf Technol 59(9–12):1073–1089
40. Wu TH, Chang CC, Yeh JY (2009) A hybrid heuristic algorithm adopting both boltzmann function and mufation operator for manufacturing cell formation problems. Int J Prod Econ 120(2):669–688
41. Wang L, Pan QK, Suganthan PN, Wang WH, Wang YM (2010) A novel hybrid discrete differential evolution a algorithm for blocking flow shop scheduling problems. Comput Oper Res 37(3):509–520
42. Li JQ, Pan QK, Liang YC (2010) An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. Comput Ind Eng 59(4):647–662
43. Wang XJ, Gao L, Zhang CY, Shao XY (2010) A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. Int J Adv Manuf Technol 51(5–8):757–767
44. Zhao F, Hong Y, Yu D, Yang Y (2013) A hybrid particle swarm optimization algorithm and fuzzy logic for processing planning and production scheduling integration in holonic manufacturing systems. Int J Comput Integr Manuf 23(1):20–39
45. Akpinar S, Bayhan GM, Baykasoglu A (2013) Hybridizing ant colony optimization via genetic algorithm for mixed-model assembly line balancing problem with sequence dependent setup times between tasks. Appl Soft Comput 13(1):574–589
46. Muller LF, Spoorendonk S, Pisinger D (2012) A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. Eur J Oper Res 218(3):614–623
47. Moradinasab N, Shafaei R, Rabiee M, Ramezani P (2013) No-wait two stage hybrid flow shop scheduling with genetic and adaptive imperialist competitive algorithms. J Exp Theor Artif Intell 25(2):207–225
48. Yun YS, Moon C, Kim D (2009) Hybrid genetic algorithm with adaptive local search scheme for solving multistage-based supply chain problems. Comput Ind Eng 56(3):821–838
49. Yildiz AR (2009) Hybrid immune-simulated annealing algorithm for optimal design and manufacturing. Int J Mater Prod Technol 34(3):217–226
50. Noktehdan A, Karimi B, Kashan AH (2010) A differential evolution algorithm for the manufacturing cell formation problem using group based operators. Expert Syst Appl 37(7):4822–4829
51. Ho WH, Tsai JT, Lin BT, Chou JH (2009) Adaptive network-based fuzzy inference system for prediction of surface roughness in end milling process using hybrid taguchi-genetic learning algorithm. Expert Syst Appl 36(2):3216–3222
52. Zhang H, Zhu Y, Zou W, Yan X (2012) A hybrid multi-objective artificial bee colony algorithm for burdening optimization of copper strip production. Appl Math Model 36(6):2578–2591
53. Yildiz AR (2013) Optimization of cutting parameters in multi-pass turning using artificial bee colony-based approach. Inf Sci 220(20):399–407

Chapter 5
Parallelization of Intelligent Optimization Algorithm

Today, different kinds of hardware for computing are more and more powerful, in accordance with large scaled complex computing tasks. From multi-core computer to clusters, various parallel architectures are developed for computing acceleration. In terms of the long time iteration and population based mechanism of intelligent optimization algorithm, parallelization is attainable and imperative in many complex optimization. Among the existing parallel methods developed for intelligent optimization algorithm, almost all of them are established upon population division with periodical communication. In several cases, the performances of different topologies and different communication mechanisms are varied. Thus in acceleration of intelligent optimization algorithm, the selection and design of topology and communication mechanism are two crucial parts and can also be configured flexibly.

That is to say, the implementation of different topology and communication mechanism can be encapsulated into modules according to different hardware architectures. These modules are independent with the operators applied in different sub-populations, thus can be reused like operators.

According to such idea, in this chapter, we firstly introduce the parallel implementation ways of intelligent optimization algorithm on different hardware architectures. Then we elaborate the typical parallel topologies based on general population division. After that, two configurable parallel ways are presented in different hardware both with module based configuration idea.

5.1 Introduction

As parallel technology continues to evolve, peta-flops parallel computers, large-scaled distributed clusters are emerging in several areas. From the perspective of computing hardware, the pure computing capabilities are largely improved.

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_5

However, the development is gradually out of Moore's law. That means, the computing speed is no longer grown with the increased computing cores. No matter in manufacturing or industrial engineering, high performance hardware not only did not bring about enough acceleration, but also induced several problems, such as load imbalance, communication blocking, etc., with large energy wasting. Therefore, the performance of parallel technology depends not only on its hardware, but also on the computing and communication assignments and the algorithm design.

In manufacturing, facing with mass productive resources and large-scaled tasks, the optimal problems such as resource scheduling, workflow arrangement and part design in diverse networked manufacturing modes are becoming more and more complex. Its evaluation indexes are generally non-linear functions, and the production steps are increased. For accelerating the whole process, the attention is gradually switching from accuracy improvement to parallelization digging. In such heterogeneous manufacturing system, the parallelization of optimization algorithms is the most important part. That is because both coarse grained and fine grained manufacturing tasks are directly scheduled and parallelized in distributed resources through different kinds of algorithms. The efficiency of task execution is decided by optimization algorithms. If we have an efficient optimization algorithm with high decision accuracy and high speed, then the whole system can be effectively accelerated.

Hence, the parallelization of intelligent optimization algorithms for diverse manufacturing optimization problems is of the essence. Back to its basic operators, according to the theory of "no free lunch" pointed out by [1], any performance improvement in algorithm needs some sacrifice from other sides. For large scaled problems, high quality solutions are obtained either by increasing iteration number and population number, or by adding other improved and hybrid operators. All of these modifications increase the time consumption in decision. Population-based parallelization can largely reduce the time consumed by the operators with high computing complexity. Its mainframe can be shown in Fig. 5.1. In such framework, more operators can be applied with lesser individuals in each sub-population. But in each sub-population, if without communication, the independent iteration will be much less efficient. That is to say, the accuracy preservation is realized by periodical individual exchange among sub-populations.

Based on the framework, parallel intelligent optimization algorithm can be designed and implemented in any multi-processor hardware with uniform communication topology and individual exchange mechanism. The whole design architecture can be represented in Fig. 5.2. With the characteristic of natural concurrency of intelligent algorithms, since the early 90th, a series of parallel intelligent optimization algorithms are proposed. Most of these algorithms are under three typical modes [2]: *master-slave mode*, *coarse-grained mode* and *fine-grained mode*.

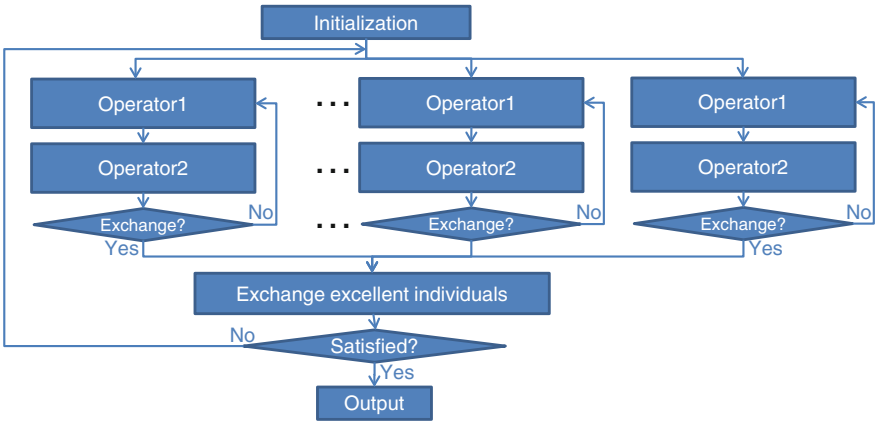


Fig. 5.1 The general framework of parallel intelligent optimization algorithm

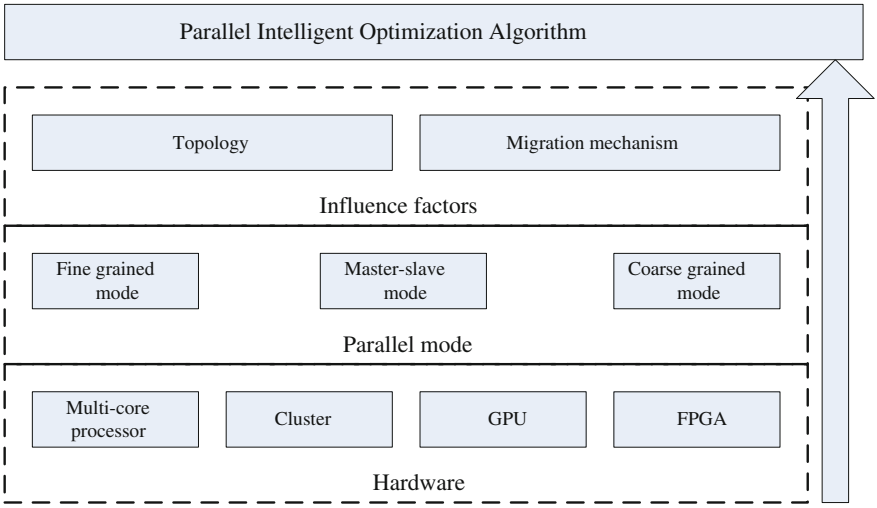


Fig. 5.2 The design architecture of parallel intelligent optimization algorithm

(1) *Fine-grained mode* [3].
In fine grained mode, each sub-population contains only one or two individuals, and operators are executed between different parallel nodes. There are no periodic exchanges but huge communications. In this mode, neighbor structure which decide the scope of learning, cross and exchange between individuals are the main consideration. It defines the information propagation path in the whole population. Generally, for small sized population, large scope structure is suitable, while for large sized population, box structure with four to eight groups are more adaptable. Shapiro et al. [4] have taken

experiments and discussions based on different neighbor structure and concluded that neighbor structure with four groups is quite adaptable.
However, fine-grained mode can only be implemented in shared memory architecture because of its huge communication load during iteration. Nowadays, it is less used.

(2) *Master-slave mode* [5].
Master-slave mode refers to use one master node to manage other slave nodes with sub-populations. Operators are executed in slave nodes and individual exchange is realized through individual reduce and broadcast by master node. Generally, in each period, slave nodes send their best individuals to master node. Master node then screen the global best one and send it again back to slave nodes as a member for next evolution. Communication mainly happens in the collection and broadcasting process and is much less than which in fine-grained mode.
This mode is still widely used for different problems, it is easy to implement and diverse population number and operators can be executed in different slave node. Uneven loads among master and slaves is the main shortcoming. In each period, the simultaneously individual sending from slave nodes can lead to data surging, and all of the slave nodes have to wait for master to calculating the best one and broadcasting it.

(3) *Coarse-grained mode* [6].
It's the most adaptable parallelization mode. Sub-populations are evenly divided and evolve independently and exchange in specific frequency. Without supervision of master node, sub-populations exchange excellent individuals in a specific topology. Communication and computation in each period are more even than master-slave mode.
In this mode, communication topology and individual migration mechanism are two main influence factors for the performance. Communication topology represents the information propagation path of each sub-group. There are already many typical topologies are presented, such as ring) [7], grid [8] and full connection and so on. In the case of sparse connection, the information transforming speed is low, and the whole population has high diversity and low collaboration. On the contrary, with dense connection, the individuals can be largely shared with low diversity and high collaboration. Premature is easily caused in such case. Therefore, the selection of topology is vital.

Besides, migration mechanism can also influence the optimal searching process. It includes the design of individual number to be shared, the period and the replacing scheme. The individual number to be shared refers to the number of individuals sending by each sub-population. The period means the generation number between two exchanges. The replacing scheme decides which local individuals are to be replaced by the newly introduced ones from other sub-groups. This three migration factors, together with topology determine the whole parallel searching performance. Matsumura et al. [9] had compared different topologies in some cases, but different migration mechanisms are still to be discussed.

Clearly, different topologies and migration mechanisms can be reused in different problems. With the idea of dynamic configuration, they can be dynamically configured in different generation or different node with different hardware. Therefore, in the sections below, we will follow Fig. 5.2 and briefly talk about different implementation ways of intelligent optimization algorithm in different hardware, and then give some typical parallel topologies commonly used in industrial optimizations. Based on that, the configuration design of parallel intelligent optimization algorithm is elaborated in this chapter.

5.2 Parallel Implementation Ways for Intelligent Optimization Algorithm

5.2.1 Parallel Implementation Based on Multi-core Processor

Multi-core processor is a processor that integrates two or more complete cores. To traditional single-core processor, the way to increase the processing speed is to improve its work frequency. However, the frequency improvement means the improvement of manufacturing process, which is not unlimited. Quantum effect largely restricts the work frequency and the size of transistor. Thus, the method to boost CPU performance by increasing processor frequency encounters an unprecedented predicament.

The emergence of multi-core processors brings new hope to the improvement of processing speed. Large companies such as INTEL and AMD in turn modify the architectures of CPU, integrate multiple cores in one chip and hence launch dual-core, three-core and four-core CPU products. Figure 5.3 shows the architecture of a multi-core processor.

Currently, based on multi-core processor, many researches [10] focus on the design of parallel programs and parallel technology (e.g. OpenMP). Among the many parallel optimization algorithms based on multi-core processor, some are implemented with the help of OpenMP techniques. In fact, as long as the programmers encode the optimization algorithms in parallel programs and run them on parallel multi-core processor computers, these parallel algorithms can be well implemented. Accordingly, Mahinthakumar and Saied [11] successfully completed the parallel Genetic Algorithm (GA) on multi-core processor while Wang et al. [12] made the parallel Particle Swarm Optimization Algorithm possible and took advantage of it to solving facility location problems. Rajendran and Ziegler [13] well introduced parallel Ant Colony Algorithm based on multi-core processor and solved permutation flowshop scheduling problem.

To sum up, many research and studies have already been conducted on the topic of multi-core processor parallel optimization algorithms, and their successful research achievements involve Genetic Algorithm, Particle Swarm Algorithm and Ant Colony Algorithm, etc.

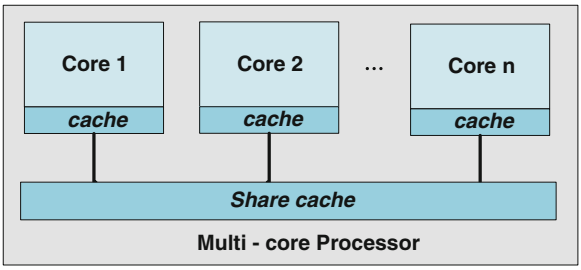


Fig. 5.3 Multi-core Processor

5.2.2 Parallel Implementation Based on Computer Cluster

Computer cluster is a special computer system with a set of loosely integrated computer software and hardware, which closely collaborates to complete the computational works efficiently. To some extent, a computer cluster can be regarded as a single host computer. Moreover, a single computer in the cluster system is often referred to as a node, usually connected via a LAN, but there are other possible connections. Computer cluster is usually used to improve the computing speed and reliability of single computer.

In a computer cluster, the multiple processors usually work in parallel, and every processor has more than one computational core. Therefore, the level of parallelism of a computer cluster is far higher than that of a multi-core processor. Usually, the design of parallel algorithms on computer clusters depends on MPI and OpenMP programming [14, 15].

In recent years, people have carried on some research about the design of parallel optimization algorithms based on computer clusters. E.g., Kalivarapu [16] thoroughly analyzed and discussed the parallel implementation method of Particle Swarm Algorithm, including the implementation on computer clusters. Also, Borovska [17] and Sena et al. [18] programmed optimization Ant Colony Algorithm on computer clusters and tested it through solving the problem of TSP. However, other types of cluster optimization algorithms are relatively less.

In short, through designing parallel optimization algorithms on computer cluster, we can acquire higher level of parallelism than on multi-core processor. Currently, the relatively mature parallel algorithms are computer cluster-based Particle Swarm Algorithm and Ant Colony Algorithm.

5.2.3 Parallel Implementation Based on GPU

Computer graphics processor (Graphics Processing Unit, GPU) is defined as ‘a single-chip processor, integrated geometric transformation, illumination, triangular configuration, clipping and drawing engine and other functions, and having

per second at least 10 million polygons handling capacity.' GPU greatly enhanced the processing speed of the computer graphics and the graphics quality, and meanwhile promoted the rapid development of computer graphics applications. Unlike the serial design pattern of the central processor (Central Processing Unit, CPU), GPU is initially designed for the graphics processing, thus, has a natural parallel character. However, the parallelizable instructions in computation are less, and increasing instruction-level parallelism through superscalar, deep water, long instruction word cannot achieve good results.

Because the graphics processor is equipped with parallel hardware structure, thus the calculation performed in the graphics processor has a natural parallelism. These years many research about the design of GPU-based optimization algorithms emerged. The research in [19, 20] studied how to implement optimization algorithms on GPU, while Kalivarapu [16] not only achieved the implementation of Particle Swarm Algorithm on computer clusters, but also on GPU. In addition, relied on GPU, Zhu and Curry [21] gave a detailed study of Ant Colony Algorithm and its parallel application, and Chitty [22], Li et al. [23] carried on a specifically concrete work on the implementation of Genetic Algorithm. There are many other literatures focused on this field of study, which have done a lot of concrete works.

In summary, nowadays, parallel GPU-based optimization algorithm design has been extensively studied and has accomplished a variety of intelligent optimization algorithms and comparatively good results.

5.2.4 Parallel Implementation Based on FPGA

Field-programmable gate array (FPGA) is a further developed product on the basis of many programmable devices such as the PAL, GAL, and CPLD. It is a semi-custom circuits,which is different from the Application Specific Integrated Circuit (ASIC).

In general, both multi-core processor and graphics processor have fixed hardware circuit structures, on which the design of algorithms fails to have a high level of parallelism. Moreover, although general-purpose microprocessors are flexible to design and easy to upgrade, their processing speed and efficiency are relatively low. On the other hand, ASIC can complete the computation tasks by a specific operation and processing unit, thus the execution of instructions is in parallel. To specific integrated circuits, the processing speed and efficiency are higher, but the development cycle is longer and the design flexibility is less. Therefore, in some occasions with higher requirements of real-time performance and flexibility, general-purpose microprocessor or ASIC is not able to solve the problem very well. FPGA, with a natural parallel hardware structure, is not only flexible to design and easy to upgrade as general-purpose microprocessor, but also faster and more efficient as ASIC. Therefore, it provides a new way for the parallel design of optimization algorithms.

So far, FPGA-based optimization algorithm parallel design is not that much, and existing research mainly concentrate on the FPGA implementation of Genetic Algorithm [24, 25] and Ant Colony Algorithm [26]. The studies on other FPGA-based optimization algorithms, such as Particle Swarm Algorithm, are far less and need to be further investigated.

Currently, people have carried out series of research about the design of parallel optimization algorithms, including Genetic Algorithm, Particle Swarm Algorithm and Ant Colony Algorithm. The common method of design is to program with the use of OpenMP. Through programming parallel optimization algorithms on computer clusters, we can obtain a higher level of parallelism than multi-core processor-based design. What's more, after the successful implementation of parallel Genetic algorithm and Particle Swarm Algorithm on computer clusters, they reap a quite extensive application, and the general pattern is to design programs by OpenMP and MPI. Until now, the most widely used parallel technology is the parallel program design based on GPU. Many intelligent optimization algorithms have been well implemented on GPU, and the results seem bright and promising. This method mainly takes advantages of the special hardware structure of GPU. Finally, studies about the algorithm design based on FPGA are chiefly focused on Genetic Algorithm. As to others types of algorithms, there is no common parallel method.

5.3 Implementation of Typical Parallel Topologies for Intelligent Optimization Algorithm

As introduced before, parallel topology represents the connection way among sub-populations. It controls the transform ways and speeds of excellent individuals, so as to make the parallel searching process exerting different influences in varied cases. During existing methods, master-slave topology and ring topology, mesh topology and full-mesh topology in coarse grained mode are the most typical ones. In this section, based on MPI architecture, we will introduce them and give brief MPI implementation of them respectively, from sparse connection to dense connection. Each of the implementation can be encapsulated as modules and reused with configuration methods.

5.3.1 Master-Slave Topology

As shown in Fig. 5.4a, the operation in slave node contains intelligent optimization operators, general evolutionary update steps and individual sending actions,

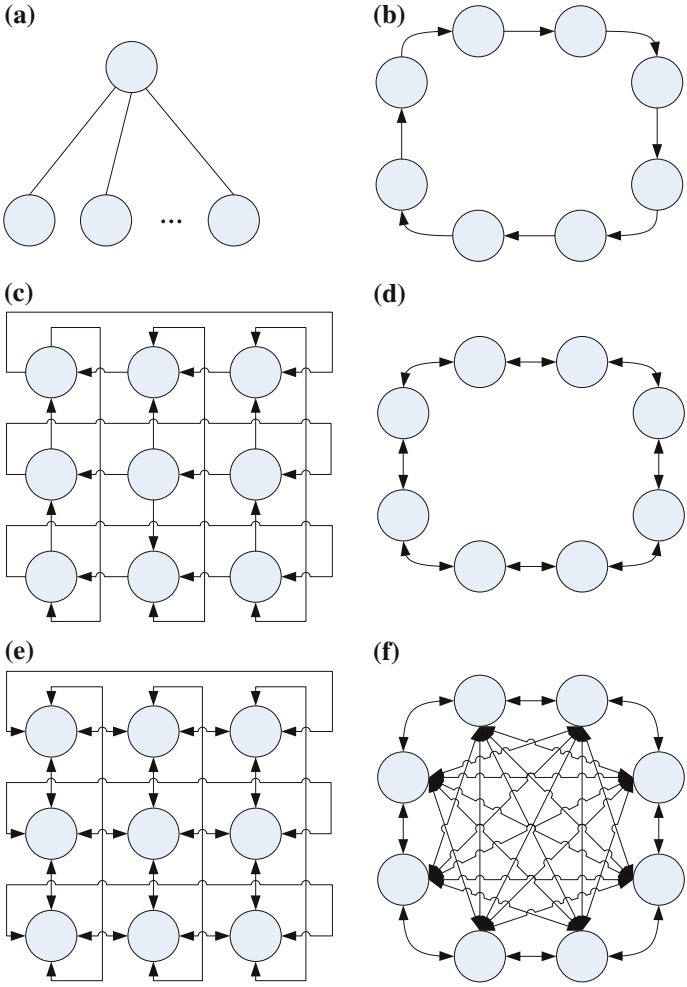


Fig. 5.4 Typical parallel topologies for intelligent optimization algorithm. **a** Master Slave; **b** Single Ring; **c** Single Mesh; **d** Double Ring; **e** Double Mesh; **f** Full Mesh.

while the operation in master node contains only receiving the individuals, calculating the best ones and broadcasting them to slave nodes.

Specifically, the implementation pseudo-code can be represented as follows.

```
For (each sub-population I)
    Initialize subpopulation;
    generation = 0;
    While (generation <= MAX_generation or convergence criterion satisfied)
        generation ++;
        MPI_Gather(best_individual, 1, gene_struct, root_population, 1, gene_struct,
        ROOT, MPI_COMM_WORLD);
        If (processor_id == ROOT)
            choose n individuals from root_population to obest[n];
        End if
        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Bcast(obest, n, gene_struct, ROOT, MPI_COMM_WORLD);
        Insert obest[n] to each sub-population;
        If (processor_id != ROOT)
            Apply algorithm's operators;
            Evaluate solutions in the sub-population;
        End if
    End while
End for
```

In the implementation, *best_individual* represents the best individual array in each sub-population to be sent in each period. *gene_struct* represent the class type of each individual, it is generated by *MPI_Type_struct* and contains gene-bits, individual states and its fitness values. *root_population* represent the received individuals in master node (i.e. root node), so that the size of this array are decided by the number of slave nodes and the size of *best_individual*. **obest**[n] then represents the screened best individuals to be broadcasted in master node. If $n = 1$, then only one individual will be selected and broadcasted to other slave nodes. So the communication load is in part decided by n.

5.3.2 Ring Topology

Ring topology consists of two kinds, single-ring and double-ring topology, as shown in Fig. 5.4b and c. Among them, single-ring topology can also be named as the least communication topology. Ring topology is easy to implement and occupies less bandwidth during communication. Information in this topology are spread slowly. In MPI programming, only point to point communication mode can

efficiently implement it. Here takes non-blocking communication as example, the specific pseudo-codes of single-ring and double-ring topology can be shown as follows.

(a) Single-Ring Topology

```
Forward propagation
For (each sub-population I)
    Initialize subpopulation;
    generation = 0;
    While (generation <= MAX_generation or convergence criterion satisfied)
        generation ++;
        If (generation % MT == 0)
            MPI_Irecv(obest, scnt, gene_struct, processor_id+1, 123,
            MPI_COMM_WORLD, &req);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id-1, 123,
            MPI_COMM_WORLD, &req2);
            Insert obest to each sub-population;
        End if
        Apply algorithm's operators;
        Evaluate solutions in the sub-population;
    End while
End for

Back propagation
For (each sub-population I)
    Initialize subpopulation;
    generation = 0;
    While (generation <= MAX_generation or convergence criterion satisfied)
        generation ++;
        If (generation % MT == 0)
            MPI_Irecv(obest, scnt, gene_struct, processor_id-1, 123,
            MPI_COMM_WORLD, &req);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id-1, 123,
            MPI_COMM_WORLD, &req2);
            Insert obest to each sub-population;
        End if
        Apply algorithm's operators;
        Evaluate solutions in the sub-population;
    End while
End for
```

(b) Double-Ring Topology

```
For each sub-population I
    Initialize subpopulation;
    generation = 0;
    While (generation <= MAX_generation or convergence criterion satisfied)
        generation ++;
        If (generation % MT == 0)
            MPI_Irecv(obest_1, scnt, gene_struct, processor_id-1, 123,
            MPI_COMM_WORLD, &req1_1);
            MPI_Irecv(obest_2, scnt, gene_struct, processor_id+1, 321,
            MPI_COMM_WORLD, &req2_1);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id+1, 123,
            MPI_COMM_WORLD, &req1_2);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id-1, 321,
            MPI_COMM_WORLD, &req2_2);
             Insert obest_1 and obest_2 to each subpopulation;
        End if
        Apply algorithm's operators;
        Evaluate solutions in the subpopulation;
    End while
End for
```

In the above codes, *obest*, *obest_1*, *obest_2* represents temporary arrays for receiving neighbor individuals, *scnt* represents the number of individuals to be migrated and *req1_1*, *req1_2*, *req2_1* and *req2_2* are *MPI_Request* parameters. It can be seen that forward propagation and back propagation ring communication can be separated as two modules and implemented as the above pseudo-code (a). Then (b) is the fusion of the two single-track communications as double-ring topology.

5.3.3 Mesh Topology

Mesh topology is similar with ring topology. It contains both left-right and up-down neighbor communication, while ring topology contains only left-right neighbor communication. If the process node is less than 9, the communication will be uneven and cause large exchange load. Also, mesh topology can be implemented as single-side topology and double-side mesh topology. The key MPI implementation can be shown as follows.

(a) Single-side mesh topology

```
For (each sub-population I)
    Initialize sub-population
    generation = 0;
    While (generation <= MAX_generation or convergence criterion satisfied)
        generation ++;
        If (generation % MT == 0)
            //left-right
            MPI_Irecv(obest_1, scnt, gene_struct, processor_id-1, 123,
            MPI_COMM_WORLD, &req1_1);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id+1, 123,
            MPI_COMM_WORLD, &req1_2);
            //up-down
            MPI_Irecv(obest_2, scnt, gene_struct, processor_id-m, 321,
            MPI_COMM_WORLD, &req1_1);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id+m, 321,
            MPI_COMM_WORLD, &req1_2);
            Insert obest_1 and obest_2 to each subpopulation;
        End if
        Apply algorithm's operators;
        Evaluate solutions in the subpopulation;
    End while
End for
```

(b) Double-side mesh topology

```
For (each sub-population I)
    Initialize subpopulation;
    generation = 0;
    While (generation <= MAX_generation or convergence criterion satisfied)
        generation ++;
        If (generation % MT == 0)
            //left-right
            MPI_Irecv(obest_1, scnt, gene_struct, processor_id-1, 123,
            MPI_COMM_WORLD, &req1_1);
            MPI_Irecv(obest_2, scnt, gene_struct, processor_id+1, 321,
            MPI_COMM_WORLD, &req2_1);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id+1, 123,
            MPI_COMM_WORLD, &req1_2);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id-1, 321,
            MPI_COMM_WORLD, &req2_2);
            //up-down
            MPI_Irecv(obest_3, scnt, gene_struct, processor_id-m, 123,
            MPI_COMM_WORLD, &req1_1);
            MPI_Irecv(obest_4, scnt, gene_struct, processor_id+m, 321,
            MPI_COMM_WORLD, &req2_1);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id+m, 123,
            MPI_COMM_WORLD, &req1_2);
            MPI_Isend(best_individual, scnt, gene_struct, processor_id-m, 321,
            MPI_COMM_WORLD, &req2_2);
            Insert obest_1, obest_2, obest_3, obest_4 to each sub-population;
        End if
        Apply algorithm's operators;
        Evaluate solutions in the sub-population;
    End while
End for
```

In such topology, the serial numbers of the neighbors at the up-down side can be calculated as the integral upper bound of the square root of the whole processor number.

$$m = \overline{\sqrt{processor\_number}} \tag{5.1}$$

5.3.4 Full Mesh Topology

In this topology, each sub-population broadcast its best individuals to be migrated. After that, each sub-population receives multiple individuals and accepts part or all of them to replace some bad local ones. It is the most communication topology and have high communication load. Compared with master-slave topology, it is more likely to cause data surging. In large-scaled distributed parallel architecture, it is not suitable. In the MPI programming, full mesh topology is easy to implement with MPI_Allgather. The pseudo-code can be represented as follows.

```
For (each sub-population I)
    Initialize subpopulation;
    generation = 0;
    While (generation <= MAX_generation OR convergence criterion satisfied)
        generation ++;
        If (generation % MT == 0)
            MPI_Allgather(best_individual, scnt, gene_struct, obest, scnt,
            MPI_COMM_WORLD);
            Insert obest[n] to each subpopulation;
        End if
        Apply algorithm's operators;
        Evaluate solutions in the sub-population;
    End while
End for
```

In the implementation, the size of **obest** n is decided by the migration number and the whole processor number.

$$n = scnt \times processor\_number \tag{5.2}$$

5.3.5 Random Topology

During the above topologies, no matter with dense or sparse connections, have advantages and disadvantages. For balance the two kinds, Defersha and Chen [27, 28] presented a new random topology for parallelization of intelligent optimization algorithm in solving manufacturing optimization. In such topology, the exchanges among sub-populations are decided by a binary matrix. It is generated by a single node and then broadcast to others to make sure the correct exchange. The dimension of the matrix is equal to the number of processor nodes. Let **A** represents the matrix. If the element $a_{ij} = 1$, then node i will send some excellent individuals to node j. The value of a_{ij} can be calculated as follow [27].

$$a_{ij} = \begin{cases} 1, & rand() < \rho \text{ and } i \neq j \\ 0, & otherwise \end{cases} \tag{5.3}$$

where 'rand()' represents random generalized number and $\rho \in [0, 1]$ refers to the control parameter of communication density. If ρ is low, the communication is becoming sparse, vice versa. At the same time, Defersha also concludes that $\rho = 0.5$ is generally suitable for individual exchanges.

In MPI implementation, only point to point mode can be applied. With such method, we found that the matrix generation and broadcasting still takes extra time consuming which can not be ignored. Therefore, its performance in different cases is still to be discussed. The pseudo-code of random topology can be shown as follows.

```
For (each sub-population I)
    Initialize sub-population;
    generation = 0;
    obest[n] = 0;
    While (generation <= MAX_generation or convergence criterion satisfied)
        generation ++;
        If (generation % MT == 0)
            Generate random_matrix[processor_number][processor_number]; //1 or 0
            For (processor_id_i = 1 to processor_number)
                For (processor_id_j = 1 to processor_number)
                    If (i != j && random_matrix[i][j] == 1)
                        MPI_Irecv(obest[i], scnt, gene_struct, j, 123,
                        MPI_COMM_WORLD, &req);
                        MPI_Isend(best_individual, scnt, gene_struct, i, 123,
                        MPI_COMM_WORLD, &req2);
                    End if
                End for
            End for
            For (k = 1 to n)
                If (obest[k] != 0)
                    Insert obest[k] to each sub-population;
                End if
            End for
        End if
        Apply algorithm's operators;
        Evaluate solutions in the sub-population;
    End while
End for
```

In the above code, **random_matrix** represents the random matrix **A**. In each period, a new **random_matrix** is generated by root node and broadcast to others. **obest** stores the individuals received from other nodes decided by the random matrix. If $\mathbf{obest}[i] \neq 0$, then insert it into the local population and replace a bad one.

Besides, more information about MPI programming can be found in [29].

5.4 New Configuration in Parallel Intelligent Optimization Algorithm

Generally speaking, in parallel searching, communication is independent with operators. As with configuration in algorithm improvement and hybridation, parallelized algorithms can also be dynamically configured in different hardware architecture. But with different communication prototypes, parallel configurations in different hardware are totally different. Therefore, this section mainly focuses on the parallelization and algorithm configuration on general multi-processors and FPGA respectively.

In large sized multi-processors, take general cluster with MPI as an example, configuration can be classified into two kinds, (1) topology configuration, and (2) operation configuration. Topology configuration refers to invoke different communication topology in different period, while operation configuration here consists of algorithm-based, operator-based and parameter-based configuration introduced in Chaps. 3 and 4. In small sized hardware, i.e. FPGA, topology configuration cannot be implemented in most time. Parallelization based on FPGA is totally different with which in other hardware. Without population division, it parallelizes operators, encapsulated them as modules and tries to flexibly connect different parts together. That is to say, the inner part of the module cannot be changed but only reloaded. Therefore, in FPGA, we could only connect different kinds of algorithm modules or operator modules in divided generations to realize flexible configuration, here we call it module-based configuration. The configuration types on the above two hardware architecture can be summarized as shown in Fig. 5.5.

Regardless of which kinds of hardware we are based, the general design process of parallel intelligent optimization algorithm can be shown in Fig. 5.6. The steps contain (1) algorithm design, (2) scale of sub-populations, (3) topology selection, (4) migration mechanism decision, and (5) algorithm implementation. If we want to design and implement a parallel intelligent optimization algorithm, we need first to design a serial algorithm with improved or hybrid operators which can solve the specific problem with high accuracy. Next, according to the existing environment, the scale of sub-populations needs to be specified before topology design, for the reason that the exchange performance of topology depends on the number of subgroups. Based on particular topology and algorithm, we could then set the migration mechanism, i.e. how many individuals to be migrated and which of the local ones to be replaced. Based on these decisive factors, the parallel algorithm can finally be implemented. Likely, general design process is quite cumbersome.

If we encapsulate these topologies with various mechanisms, combined with the above-mentioned serial algorithm parts, the design of parallel intelligent optimization algorithm can be easier. Figure 5.7 shows the new configuration process for it. In this mode, the topology module can be invoked as a function which only implementing data transform and individual replacement. Take ring topology as an example, the corresponding module mainly contains the following part. That is to

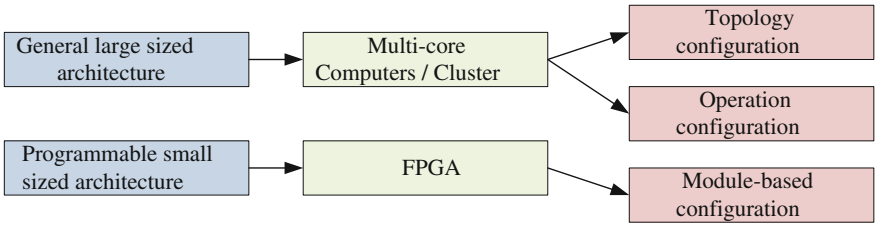


Fig. 5.5 parallel configuration types in different hardware

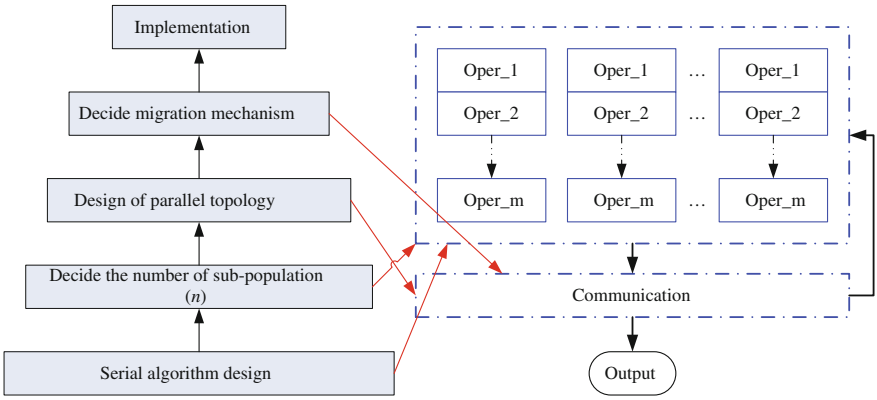


Fig. 5.6 The general design process of parallel intelligent optimization algorithm

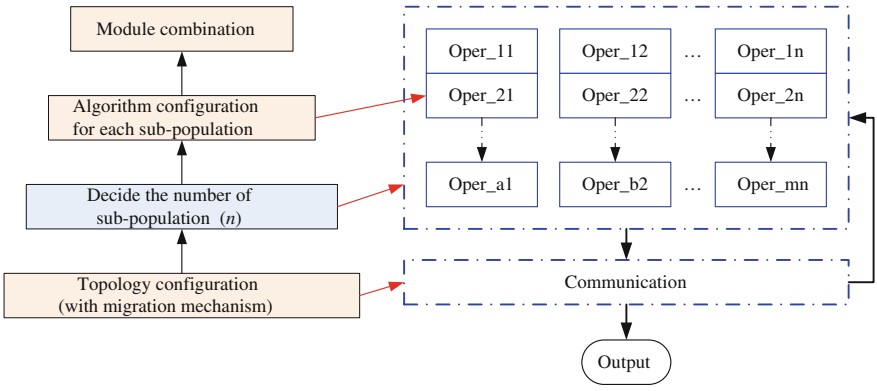


Fig. 5.7 Configuration process of parallel intelligent optimization algorithm

say, we just put data sending, receiving and the individual replacement sentences into the topology module. The parameters of it include sending array, receiving array and number of individuals to be migrated. In each sub-population, basic operators are invoked in every generation, while topology module is called at a certain period.

```
//Topology module
Basic_ring_topology(best_individual, obest, scnt)
{
    MPI_Irecv(obest, scnt, gene_struct, processor_id+1, 123, MPI_COMM_WORLD,
    &req);
    MPI_Isend(best_individual, scnt, gene_struct, processor_id-1, 123,
    MPI_COMM_WORLD, &req2);
    Insert obest to each sub-population;
}

//Module Invoking
If (generation % MT == 0)
    Basic_ring_topology(best_individual, obest, scnt);
End if
```

It is clear that in traditional design ways, the selection of topology is dependent with both the design of serial operators in each sub-group and the hardware environment. If any of them performs not well, then we need to redesign it again. Different with the traditional process, we could select the topology firstly only according to the specific hardware environment. Then in each sub-group, different operators with uniform population input and output can be tested and applied respectively. In such case, topology is independent with operators. The only thing we need to do is module combination. With different operators, the sub-group who performs better could help other bad performed ones to break out from local optimal through individual exchange. Then better optimal searching capability can be preserved with low time consumption for wider complex problems.

In the following sections, we will elaborate different configuration types both in multi-processor computers and FPGA.

5.4.1 Topology Configuration in Parallelization Based on MPI

As mentioned previously, topology configuration means to apply multiple topologies in a parallel algorithm. It also contains two styles, (1) single-domain topology configuration and (2) multi-domain topology configuration.

Firstly, single-domain topology configuration is to allocate multiple topologies into different generations with one operation domain, as shown in Fig. 5.8. In such scheme, although operators in different sub-population are different, they will not change along with iteration. All sub-populations belong to one domain. After dividing the generations into several parts, we could change topology module to

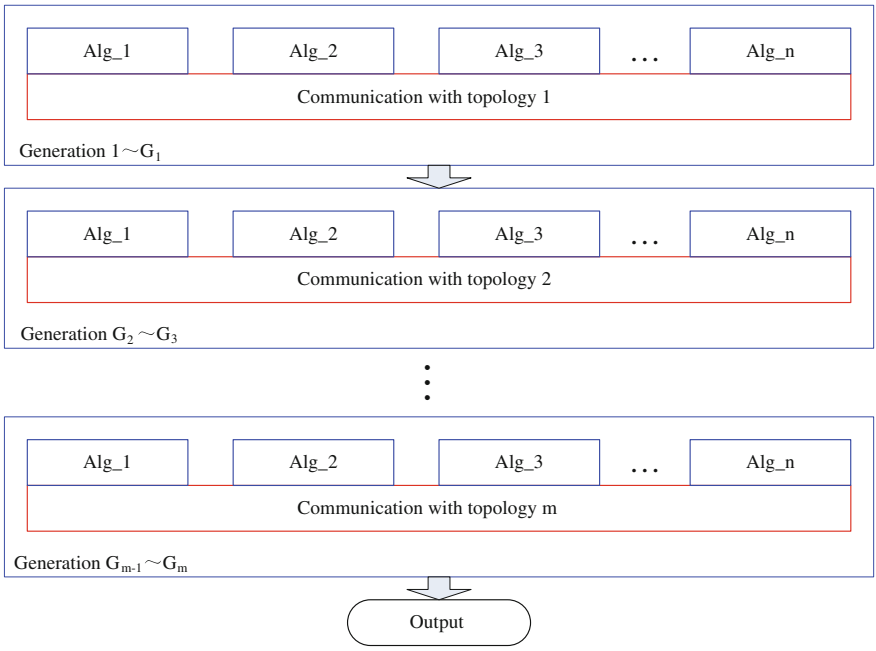


Fig. 5.8 Single-domain topology configuration in parallel intelligent optimization algorithm

make sub-populations communicating with different ones, as well as in random topology. Moreover, extra communication is needless. The information propagation can be easily controlled according to the whole population state. If the population has high diversity, then the topology with dense connection can be applied. On the contrary, if the population has low diversity, then the topology with sparse connection is more suitable. Operators in different sub-groups are responsible for digging solution with lesser members and less time, topology then tries to balance the searching state and preserve high quality. Following the general searching rules, topology with sparse connection should be adopted at the beginning for dynamic exploration. Then topology with dense connection can be used in the end for population convergence accordingly.

Multi-domain topology configuration, as shown in Fig. 5.9, refers to divide sub-populations into several domains and apply different topology to each domain. In each period, sub-population with different algorithms only communicates with the ones in the same local area through corresponding connection topology. Groups in different domains will not do exchange any more. For wider information exchange, we could also divide generations into several parts and regroup the sub-populations to different topology domains. It is clear that the information propagation is narrower and slower than which in single-domain scheme. It can keep better diversity state and is more suitable for heterogeneous clusters, in which we could allocate sparse topology to the nodes with low communication bandwidth and dense

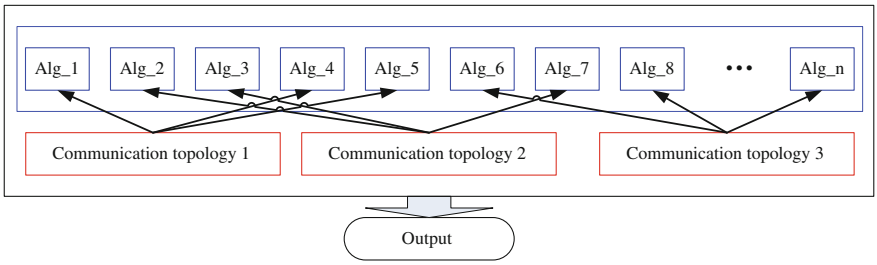


Fig. 5.9 Multi-domain topology configuration in parallel intelligent optimization algorithm

topology to the nodes with high communication speed. In this scheme, one of the most important steps is regrouping. We could generate a group of random numbers which refer to the topology numbers in a root node and then broadcast them for allocating sub-populations uniformly. So, the main drawback of this method turns out to be the restructuring step which may take many extra times so as to slow down the whole process. For simplifying the process, people can also configure the same algorithm for each sub-population with only topology hybridization.

It can be seen that topology configuration is suitable especially for large scale parallelization with a large number of computing nodes. When sub-populations are less, multi-topologies are then becoming useless. For example, in a parallel intelligent optimization algorithm, if there are only four nodes (processors), then mesh topology has no much difference with full-mesh topology. The control of changing topology in single-domain scheme and the regrouping step in multi-domain scheme are both time consuming with low-efficiency.

5.4.2 Operation Configuration in Parallelization Based on MPI

Correspondingly, in small scale parallelization, operation configuration is more suitable. As mentioned before, operation configuration means to do three-layer configurations in each sub-population without topology changing. It is the same with the parallel configuration in module-based improvement and hybridization mentioned in Chap. 4. The only difference is that different operators are simultaneously executed in multiple processors. It is much easier to design than the above topology configuration ways. With fixed individual exchanging scheme, the evolutionary process is more stable.

With limited computing resources, operation configuration in parallel intelligent optimization algorithm can be very adaptable especially for dynamic complex problems, such as parameter adjustment in part design and dynamic job-shop scheduling. For instance, for a continuous parameter setting problem in part design, we could divide the whole population into four groups and applied

continuous genetic algorithm (GA), particle swarm optimization (PSO), differential evolution (DE) and cuckoo search (CS) in sub-populations respectively. According to 'no free lunch theory', these four algorithms are suitable for different cases. In simultaneously execution with exchanges, the most suitable one in a specific case will offer its current best solution to others and guide them to better positions. It the constraints are changed or a new part is needed to be designed, another algorithm might be a new leader to preserve the whole searching quality. We need not to design new algorithms, only one scheme with several configured algorithms can applied to different kinds of problems with good quality. For achieving such performance, we should note that the algorithms configured in different sub-populations need to be very different with diverse emphasis on exploration and exploitation, as well as in the design of serial intelligent optimization algorithm, for balance searching.

Moreover, it should be note that even in large-scaled parallelization design, we need not to configure topology and operators both. That is because too much dynamics will totally break the searching paces and obtain a chaos situation as a result. Therefore, although configuration is easy to implement, the collaboration between operators and topologies need to be considered seriously.

5.4.3 *Module Configuration in Parallelization Based on FPGA*

In this section, we presented a new parallelization way of intelligent optimization algorithm on FPGA. With several blank logical resources in FPGA, we could implement the original operators as multiple arithmetic units. For connecting them, some state machine is also designed for connecting these units to form a specific intelligent optimization algorithm. Based on these design structure, we will implement some typical intelligent optimization algorithm on FPGA and do some configuration design further in the following chapters.

All of the designs are established based on VHDL (Very-high-speed Hardware Description Language). It is used to describe a digital system, including its structure, behaviors, functions and interfaces. The style and grammar of VHDL is much similar to advanced computer programming languages, except that it stands for hardware describing. In VHDL, a digital system is called an entity which can be defined as inner processors and external interfaces separately. After external interfaces are set up, inner processes can be developed in detail. And then, the developed entity can be called as a subsystem in order entities. In other to help readers to learn about the design and validation of intelligent optimization algorithm on FPGA, we will introduce the Virtex-5 family FPGA chips and floating-point format of IEEE 754 in following.

Firstly, the algorithms designed in this book are implemented and validated withVirtex-5 family FPGA chips. Figure 5.10 shows their structure. It can be seen

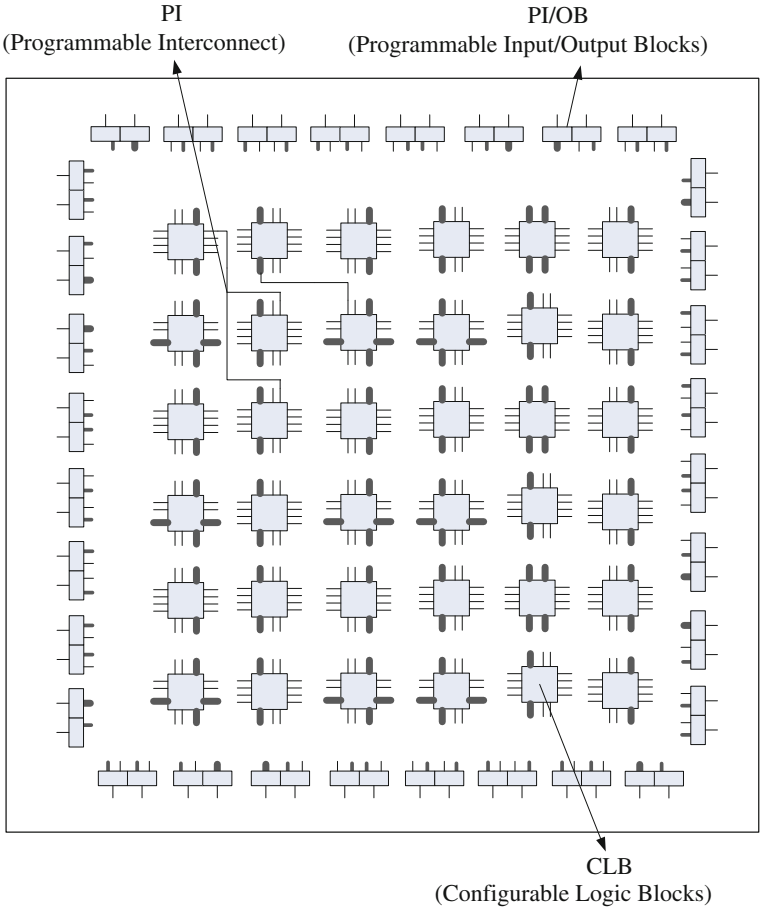


Fig. 5.10 The inner structure of FPGA

that there are mainly CLB (Configurable Logic Blocks), PI (Programmable Interconnection) and PI/OB (Programmable I/O Blocks) inside an FPGA chip. Except these three components, there are also some other abundant resources, such as DSP48E for computing, Block RAM for data storage and CMT (Clock Management Tiles) for clock managing and so on.

A CLB contains several logical resources inside, which are used to implement combinational circuit and sequential circuit. Each CLB in Virtex-5 includes 2 slices, 8 LUT (Look Up Table), 8 triggers, 2 arithmetic and carry chains, 256-bits distributed RAM and 128-bits shifting register. DSP48E Slice module in it can handle 25×18 complement multiplication and can also configured as multiplier, subtracter or accumulator. In the design process of parallel intelligent optimization algorithm, large amount of computing tasks can be assigned to this module to execute.

For different types of chips, the inner resources are different. In Virtex-5 family, there are five platforms, i.e. LX, LXT, SXT, FXT and TXT. LX and LXT are mainly used for high-speed logical design, while SXT is primarily applied for complex digital signal processing. The embedded PowerPC processor of FPGA in FXT is chiefly designed for the development of embedded system. And the FPGA of TXT is especially for customized and complete high-performance system. On account of the abundant logistical resources in FPGA, we mainly considered to use the FPGA of LX/LXT to design parallel intelligent optimization algorithms especially for the complex problems with high requirements on real-time decision efficiency. Here we list the properties of the FPGA chip of Xilinx Virtex-5 LX platform [30].

From Table 5.1 we can see that XC5VLX50T type FPGA includes $120 \times 30 = 3600$ CLBs. In the device, there are 48 DSP48E slice modules which can realize high speed floating point arithmetic together with abundant logistical resources. The block RAM which is 120×18 Kb can store plenty of intermediate data during algorithm execution. And the CMT which contains 6 time management modules is fully enough for counting the optimization time. Therefore, XC5VLX50T can fully satisfy the design requirements of intelligent optimization algorithms.

Secondly, the data format adopted by the research is floating-point data format specified by IEEE 754 standard. The standard divides floating-point data into three types: single float, double and extended. It includes three sections in the memory: sign, exponent and mantissa. For different types of floating-point data, the word lengths of the three sections are different, as shown in Table 5.2.

According to variable symbols shown in Table 5.2, a floating-point data can be calculated by the following equation.

$$x = (-1)^S \times 1.M \times 2^{E-B} \tag{5.4}$$

In the following section, for simplicity, only single-precision floating-point data format is adopted in our design of FPGA-based intelligent optimization algorithm. One must notice that the design is not limited in single float precision.

(1) **Traditional design process of parallel intelligent optimization algorithms** Multi-core processors, as well as GPU, have their own computing architecture. The processing element has its specific arithmetic unit and controller. With these units, general design process of parallel intelligent optimization algorithm can be abstracted and summarized as follows.

Step 1 **Analysis of algorithm parallelization:** In this step, we need to extract the parts which can be parallel implemented. Unlike the parallelization in coarse-grained hardware architecture, in such a fine grained chip, the cyclic parts in operators of intelligent optimization algorithm in which the data is processed independently can always be parallelized directly.

Table 5.1 Some properties of the FPGA in Virtex-5 LX platform

Device	CLB			DSP48E Slice	Block RAM			CMT
	Array (R × C)	Virtex-5 Slice	Distributed RAM(Kb)		18 Kb	36 Kb	Max (Kb)	
XC5VLX30	80 × 30	4,800	320	32	64	32	1,522	2
XC5VLX50	120 × 30	7,200	480	48	96	48	1,728	6
XC5VLX85	120 × 54	12,960	840	48	192	96	3,456	6
XC5VLX110	160 × 54	17,280	1,120	64	256	128	4,608	6
XC5VLX155	160 × 76	24,320	1,640	128	384	192	6,912	6
XC5VLX220	160 × 108	34,560	2,280	128	384	192	6,912	6
XC5VLX20T	60 × 26	3,120	210	24	52	26	936	1
XC5VLX30T	80 × 30	4,800	320	32	72	36	1,296	2
XC5VLX50T	**120 × 30**	**7,200**	**480**	**48**	**120**	**60**	**2,160**	**6**
XC5VLX85T	120 × 54	12,960	840	48	216	108	3,888	6
XC5VLX110T	160 × 54	17,280	1,120	64	296	148	5,328	6

Table 5.2 IEEE 754 standard floating-point format

Data type	Memory bits			Word length (bits)	Offset (B)
	Sign (S)	Exponent (E)	Mantissa (M)		
Single float	1(Highest)	8(23–30)	23(0–22)	32	127
Double	1(Highest)	11(52–62)	52(0–51)	64	1023
Extended	1(Highest)	15(65–79)	65(0–64)	80	16383

Step 2 **Parallel programming:** It refers to rewrite the algorithms into the corresponding parallel programming language. This step is quite related to specific processor type and the whole execution environment.

Step 3 **Debugging and improvement:** The performance of the same parallel program in different multi-core processors is varied. So the parallel codes need to be modified and improved and generate different version for users to apply.

(2) **New design process of parallel intelligent optimization algorithms on FPGA**

As we introduced before, FPGA is a kind of blank processor. It has no fixed computing architecture, no specific arithmetic unit and controller. Only a group of programmable logistical resources and other assistant resources are provided. However, general algorithm is composed by some basic calculations and process control statements. Therefore, the design of parallel intelligent optimization algorithm in FPGA is different from the above process. Extra design of relative operational unit and state machine for specific algorithm is quite essential. Then, the FPGA-based design of parallel intelligent optimization algorithm can be drawn as the following four steps.

Step 1 **Analysis of algorithm parallelization:** Firstly, all key basic calculation parts need to be listed and the parts which can be parallelized should also be extracted.

Step 2 **Design of arithmetic units:** According to the key calculation parts extracted from step 1, we need to design arithmetic units with different functions. Such arithmetic units can either be float-pointing calculation, or be binary-sequence processor. In this step, the parts which can be parallelized are implemented in the chip with multiple logistical resources. The design of calculation units combined with these resources will execute the corresponding operation in parallel. So the parallel design of arithmetic units can largely decide the whole execution performance of the algorithm.

Step 3 **Design of state machine:** In accordance with the execution process of the algorithm, we could introduce a state machine to connect and control these units for iterative searching. Each state can be seen as an executor of different calculation process. The states can be triggered in parallel.

Step 4 **Debugging and improvement:** No matter the arithmetic units or the state machine designed in FPGA for a specific intelligent optimization algorithm require some modifications in different implementation environments. And for further configuration in solving different problems, several improved versions for both the arithmetic units and the state machine are also required.

5.5 Summary

This chapter systematically introduces almost all kinds of parallel ways of intelligent optimization algorithms. Firstly, the implementation of several kinds of topologies can be applied in many areas for solving complex problem by multi-core computing resources. With these topologies, a lot of parallel intelligent optimization algorithms can also be generated quickly. And through generation division and population division, configuration can also be flexibly implemented in parallel intelligent optimization algorithm. This chapter presented two kinds of configuration ways for fully using existing algorithms and solving wider problems with complex properties. Moreover, the parallelization way of intelligent optimization algorithms based on FPGA is also introduced.

Of cause all of these configurations are not only based on the establishment of a group of typical operators and algorithms but also rely on the implementation of typical topologies in coarse-grained hardware architecture and the design of arithmetic units and state machines in fined-grained FPGA platform. But in diverse parallel platform we must know that not the design of operator but the design of parallel structure and the flexible configuration in different structure worth more in solving large-scaled optimization algorithms. As a group of new design schemes, the parallelization of intelligent optimization algorithm based on the concept of DC-IOA can be applied to reconnect existing operators or algorithms to solve wider problem more conveniently and more easily.

References

1. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
2. Crainic TG, Toulouse M (2010) Parallel meta-heuristics. Gendreau M, Potvin J-Y (eds) Handbook of metaheuristics, vol 146, pp 497–541
3. Collins RJ, Jefferson DR (1991) Selection in massively parallel genetic algorithms. In: The international conference on genetic algorithms
4. Shapiro BA, Wu JC, Bengali D, Potts MJ (2001) The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation. Oxford University Press, Oxford
5. Sun D, Sung WP, Chen R (2011) Master-slave parallel genetic algorithm based on MapReduce using cloud computing. Appl Mech Mater 121–126:4023–4027

6. Lin SC (1994) Coarse-grain parallel genetic algorithms: categorization and new approach. In: The 6th IEEE symposium on parallel and distributed processing, pp 28–37
7. Beckers MLM, Derks EPPA, Melssen WJ, Buydens LMC (1996) Using genetic algorithms for conformational analysis of biomacromolecules. Comput Chem 20(4):449–457
8. Fukuyama Y, Chiang HD (1996) A parallel genetic algorithm for generation expansion planning. IEEE Trans Power Syst 11(2):955–961
9. Matsumura T, Nakamura M, Okech J, Onaga K (1998) A parallel and distributed genetic algorithm on loosely-coupled multiprocessor system. IEICE Trans Fundam Electron Commun Comput Sci 81(4):540–546
10. Akhter S, Roberts J (2006) Multi-core programming. Intel Press, Hillsboro
11. Mahinthakumar G, Saied F (2002) A hybrid MPI-OpenMP implementation of an implicit finite-element code on parallel architectures. Int J High Perform Comput Appl 16(4):371–393
12. Wang D, Wu CH, Ip A, Wang Q, Yan Y (2008) Parallel multi-population particle swarm optimization algorithm for the uncapacitated facility location problem using openMP. IEEE Congress Evol Comput 1214–1218
13. Rajendran C, Ziegler H (2004) Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. Eur J Oper Res 155(2):426–438
14. Dolbeau R, Bihan S, Bodin F (2007) HMPP: a hybrid multi-core parallel programming environment. In: Workshop on General purpose processing on graphics processing units (GPGPU)
15. Rabenseifner R, Hager G, Jost G (2009) Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In: The 17th Euromicro International conference on parallel, distributed and network-based processing, pp 427–436
16. Kalivarapu VK (2008) Improving solution characteristics of particle swarm optimization through the use of digital pheromones, parallelization, and graphical processing units (GPUs). Iowa State University, Iowa
17. Borovska P (2006) Solving the travelling salesman problem in parallel by genetic algorithm on multicomputer cluster. In: International Conference on computer systems and technologies, pp 1–6
18. Sena GA, Megherbi D, Isern G (2001) Implementation of a parallel genetic algorithm on a cluster of workstations: traveling salesman problem, a case study. Future Gener Comput Syst 17(4):477–488
19. Zhou Y, Tan Y (2009) GPU-based parallel particle swarm optimization. In: IEEE Congress on Evolutionary computation, pp 1493–1500
20. Mussi L, Nashed YSG, Cagnoni S (2011) GPU-based asynchronous particle swarm optimization. In: Proceedings of the 13th ACM annual conference on Genetic and evolutionary computation, pp 1555–1562
21. Zhu W, Curry J (2009) Parallel ant colony for nonlinear function optimization with graphics hardware acceleration. In: The IEEE International conference on systems, man and cybernetics, SMC, pp 1803–1808
22. Chitty DM (2007) A data parallel approach to genetic programming using programmable graphics hardware. In: Proceedings of the 9th ACM annual conference on Genetic and evolutionary computation, pp 1566–1573
23. Li JM, Wang XJ, He RS, Chi ZX (2007) An efficient fine-grained parallel genetic algorithm based on gpu-accelerated. In: 2007 NPC workshops IFIP International conference on network and parallel computing, IEEE, pp 855–862
24. Graham P, Nelson B (1996) Genetic algorithms in software and in hardware-a performance analysis of workstation and custom computing machine implementations. In: IEEE symposium on FPGAs for Custom computing machines, pp 216–225
25. Shackleford B, Snider G, Carter RJ, Okushi E, Yasuda M, Seo K, Yasuura H (2001) A high-performance, pipelined, FPGA-based genetic algorithm machine. Genet Program Evolvable Mach 2(1):33–60
26. Juang CF, Lu CM, Lo C, Wang CY (2008) Ant colony optimization algorithm for fuzzy controller design and its FPGA implementation. IEEE Trans Industr Electron 55(3):1453–1462

27. Defersha FM, Chen M (2008) A parallel genetic algorithm for dynamic cell formation in cellular manufacturing systems. Int J Prod Res 46(22):6389–6413
28. Defersha FM, Chen M (2009) A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. Int J Adv Manuf Technol 49(1–4):263–279
29. Pacheco PS (1997) Parallel programming with MPI. Morgan Kaufmann, San Francisco
30. Jose S (2006) http://www.xilinx.com/prs_rls/2006/silicon-vir/0658lxship.htm

Part III
Application of Improved Intelligent Optimization Algorithms

Dynamic configuration is designed to provide some new ways to generate more flexible and robust intelligent optimization algorithm. With different structure of configuration, most of intelligent optimization algorithms can be extended and regenerated for high efficient decision in wider range of complex problems. In the previous chapters, we have elaborated its principle and implementation ways. Some preliminary tests and the extension frameworks of DC-IOA in the improvement, hybridization and parallelization of intelligent optimization algorithm are introduced.

Except for the establishment of a general intelligent optimization algorithm library and the combinations of existing algorithms, the concept of dynamic configuration and its structures can also be used in design a single algorithm for specific problem. Therefore, in this part, we will take some typical manufacturing related problems as examples and design some new improved algorithms for them from the perspectives of algorithm improvement, hybridization and parallelization respectively.

This part includes six chapters, Chaps. 6–11. In Chaps. 6 and 7, two improved algorithms based on the idea of DC-IOA are designed and encapsulated for partner selection problem and cloud service scheduling problem in cloud manufacturing respectively. In Chaps. 8 and 9, two hybrid algorithms based on DC-IOA are also implemented for computing resource allocation problem and service dual scheduling problem in distributed integrated manufacturing system respectively. In Chaps. 10 and 11, the new FPGA-based configurable algorithms are applied for complex numerical benchmarks and the typical traveling salesman problem. The performances of the above algorithms are shown in different cases in these chapters.

Chapter 6
GA-BHTR for Partner Selection Problem

In this chapter, GA-BHTR (genetic algorithm maintained by using binary heap and transitive reduction) [1] for addressing partner selection problem (PSP) in a virtual enterprise [2] is introduced. Based on ordinary initialization, an improved binary heap strategy is configured before it with uniform population input and output to realize initialization improvement. It is designed to simplify the directed acrylic graph that represents the precedence relationship among the subprojects in PSP and enhance the searching diversity of the algorithm. Then, in order to avoid solutions from converging to a constant value early during evolution, multiple communities are used instead of a single community in GA-BHTR. Operators are configured in different communities independently. Communication among communities is executed by periodic interchange.

6.1 Introduction

With economic and market globalization and rapid development and application of Internet and information technologies, the product upgrading becomes faster and faster, and the market requirements become more and more uncertain and personalized. Under this condition, the development of new product becomes a key element for enterprise to keep core competitive advantage [3–5], especially in the electronic product market. It is impossible for a single enterprise to meet the rapidly changing marketing requirement by only integrating the resources inside. An enterprise survives by having good contacts with other enterprises or companies who hold complementary assets or products [6].

Virtual enterprise (VE) has been viewed as an effective organizational mode to solve the above problems. A VE is a temporary enterprise alliance which can meet the market requirements with low cost, high quality, quick responsiveness and more customer satisfaction and adapt to the rapidly changing environment [7–9].

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_6

The core idea of VE is to organize different enterprises into a logical alliance, in which members can collaborate with each other to implement full sharing of their manufacturing resources and capabilities (including skills, technologies, resources, data, information, and knowledge). More importantly, effective resource sharing can not only deal with the specific fast changing requirements, but also exploit market opportunities as well [10, 11].

Specifically, the life cycle of a VE has the following four stages [12–14]:

(1) *Creation*: When an enterprise wins a contract for a large project and it is unable to complete it with its own finite capacity, it searches for potential partners and negotiates with them through the information infrastructure, and then a VE is created.
(2) *Operation*: After the contracts signing among partners, the VE manages the manufacturing process or the implementation of the project.
(3) *Evolution*: The VE in this stage can be reconfigured or adjusted to meet the resource requirements when the project has changed.
(4) *Dissolution*: When the project is fulfilled, the VE is finally dissolved.

The first stage, i.e., the selection of the appropriate partners when a VE is to be established, is known as the Partner selection problem (PSP). It is a fundamental and crucial issue for the success of VE.

In order to highlight the innovation works and significance of this chapter, the related works on PSP are investigated from the aspects of (a) research filed or content of PSP, (b) the criteria or attributes considered in PSP, and (c) the approaches or algorithms for addressing PSP.

In terms of research fields or content, the existing studies on partner selection primarily focus on the following aspects: (a) partner selection for virtual enterprises [9, 13, 15, 16, 17, 18], (b) partner selection for dynamic alliances [3, 19] or strategic alliances [8, 20, 21, 22], (c) partner selection for cooperative wireless networks [23] and innovation networks [24], (d) partner selection for international joint ventures [14, 25], (e) partner selection for production networks [26, 27], and (f) supplier selection in supply chains [22, 28].

Furthermore, there are many attributes or criteria which have been considered in partner selection in different studies, these attributes are primarily classified into the following six categories:

(1) *Risk-related attributes*, including political stability, regional economy status, financial health, market fluctuations, competency, due date and performance, etc.
(2) *Cost-related attributes*, including material cost, financial cost, transportation (or logistics, delivery) cost, operational cost, and so on.
(3) *Time-related attributes*, including reaction time, processing time, completion time, due date, efficiency, delivery (or transportation) time, time sequence, project (or task) duration, etc.
(4) *Quality-related attributes*, including technology level, service level, management level, performance, and so forth.

(5) *Reliability-related attributes*, including past performance, financial status, enterprise image, cooperation history, trust, credit, reputation, etc.
(6) *Other attributes*, such as capacity resources, enterprise size, organization structure, etc.

On the other hand, many algorithms and approaches have been proposed for addressing PSP, the existing methods can be classified into the following four categories based on the works of [3, 11, 29].

(1) *Mathematical programming and modeling approaches* [18, 25, 28, 30], including integer programming [13] and their enhanced types [18].
(2) *Rating/linear weighting approaches* [14, 15, 19, 20, 31].
(3) *Fuzzy decision and multiple attribute decision-making (MADM) approaches* [3], including technique for order preference by similarity to ideal solution (TOPSIS) method [9, 17], analytic network process (ANP) [8, 21], analytical hierarchy process (AHP) [32, 33], fuzzy preference programming [15], and fuzzy set theory.
(4) *Optimization approaches and artificial intelligence techniques*, including (a) deterministic algorithms, such as the Branch and Bound algorithm [16, 34]; (b) heuristic and meta-heuristic algorithms, such as genetic algorithm (GA) [34, 35], particle swarm optimization (PSO) [6, 15, 36], Tabu Search [30, 37], ant colony optimizer (ACO) [11], quantum multi-agent evolutionary algorithm (QMAEA) [38, 39], artificial intelligence techniques [27] and intelligent optimization algorithms [40].

However, when using approximate algorithms such as genetic algorithm (GA) to solve PSP, the following problems are often encountered:

(1) The first problem is that the algorithms usually converge quickly to a local best solution. The reason is that during the evolution, the algorithms would make all the solutions converge to a relatively better solution which is not the best global solution. As a result, the algorithms would seldom or never reach the final best solution.
(2) The second problem is that the operations are time-consuming. To keep the volume of the community at a constant value, many inferior individuals should be "killed" according to their fitness value. Killing individuals is a very time-consuming job. It is therefore difficult to select the best individuals from a large number of candidates.
(3) The third problem is that the precedence relationship among the subtasks or the subprojects in a PSP is too complex, and few attempts have been made to simplify the topology of the established directed acrylic graph according to the precedence relationship. Much computational effort is usually wasted on some unnecessary relationships.

In order to address the above problems, a genetic algorithm maintained using the binary heap and transitive reduction (denoted as GA-BHTR for simplification)

is designed in this chapter for addressing some specific PSP issues. The main idea of this chapter is as follows.

- In the proposed GA-BHTR, the directed acrylic graph of the specific PSP is simplified using transitive reduction in order to reduce the complexity and the computational load of the algorithms.
- In order to avoid solutions from converging to a constant value early during evolution, operators are configured in multiple communities instead of a single community. The method and algorithms for distributing the individuals to the multiple communities, and maximizing the differences among different communities are proposed in detail.
- Catastrophe operator is introduced in the proposed GA-BHTR in order to avoid the solutions from converging to a local best solution too early after many generations of evolution.
- In order to maintain the capacity of the community, i.e., the number of individuals existing in a community, in a constant value while enhancing the diversity of the proposed GA-BHTR, an algorithm using the binary heap to maintain the data is designed.

6.2 Description of Partner Selection Problem in Virtual Enterprise

6.2.1 Description and Motivation

In this chapter, the popular product iPhone4 is taken as an example to demonstrate PSP in VE. The main parts of iPhone4 are shown in the first and third column of Table 6.1. And the suppliers of these parts are listed in the second and forth column. The ones with "$\surd$" are the suppliers chosen by Apple Company. Figure 6.1 shows the parts of iPhone4 and its suppliers.

From Fig. 6.1, it can be concluded that most parts of iPhone4 are provided by other companies located in different countries. The whole production process of iPhone4 is completed by its business partners. Apple Company is only responsible for the researches of key technology and design of its products. During the production process, Apple Company does not need to hire workers and buy production line to produce the products or parts. What does Apple Company do is just selecting corresponding part suppliers and subcontracting production tasks selected companies. It makes the process more flexible. Therefore, Apple Company can focus on the development of new products. The manufacturing of iPhone4 is a typical application of VE model.

Apparently, in the process, the selection of part suppliers becomes an important problem for Apply Company, the lead enterprise in the VE. Take DRAM memory and flash memory chip as an example, many companies like LG, Samsung, Intel,

Table 6.1 Main parts and the supplier partners of iPhone4

Parts	Supplier	Parts	Supplier
LCD display	LG √	Flash memory chip	LG
	Samsung		Samsung √
	Intel		Intel
	Broadcom		Broadcom
	Texas instruments		Texas instruments
	TOSHIBA		TOSHIBA
	Motorola		Motorola
	Infineon		Infineon
Application processor	LG	DRAM memory	LG
	Samsung √		Samsung √
	Intel		Intel
	Broadcom		Broadcom
	Texas instruments		Texas instruments
	TOSHIBA		TOSHIBA
	Motorola		Motorola
	Infineon		Infineon
WIFI, Bluetooth, GPS chips	LG	Touch-screen control	LG
	Samsung		Samsung
	Intel		Intel
	Broadcom √		Broadcom
	Texas instruments		Texas instruments √
	TOSHIBA		TOSHIBA
	Motorola		Motorola
	Infineon		Infineon
Radio frequency memory	LG	Receiver/transceiver	LG
	Samsung		Samsung
	Intel √		Intel
	Broadcom		Broadcom
	Texas instruments		Texas instruments
	TOSHIBA		TOSHIBA
	Motorola		Motorola
	Infineon		Infineon √

Broadcom, Texas instruments, TOSHIBA, Motorola and Infineon can provide these two parts for Apple Company. Apple Company decides to choose Samsung to supply the DRAM memory and flash memory chip according to its own criteria. Bad decision in partner selection will lead to low-quality products, lower profits

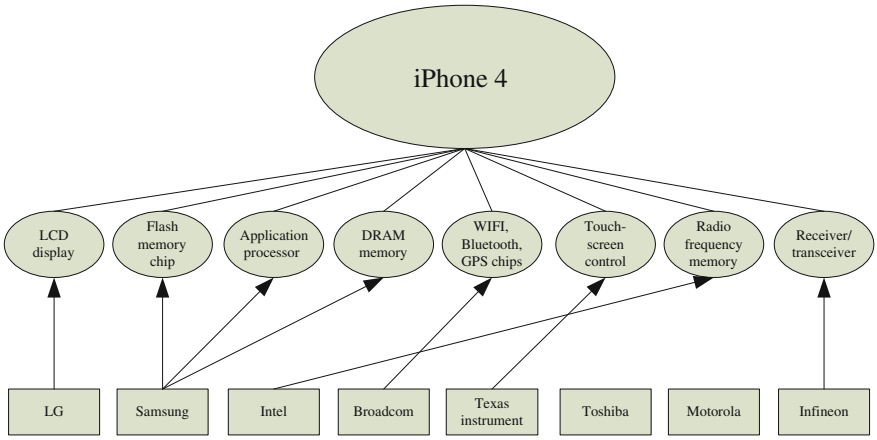


Fig. 6.1 PSP in iPhone4

and bad market reputation. For example, in 2010, the iPhone4 was out of stock for a period of time. It was hard for people to buy an iPhone4, and the customers had to wait for a long time to get one after ordering it from the online apple store. Why did that happen? Because there was something wrong with the parts' supplier partner selected by Apple Company and the supply of the LCD can't meet the requirements. Obviously, this problem had caused significant loss to Apple Company. If Apple Company selected the suppliers with more suitable capabilities, the out-of-stock of iPhone4 can be avoided. After the launch of iPhone4, there is often news that the production of iPhone4 parts severely pollutes the environment, which causes a certain degree of image damage to Apple Company. This suggests that during the partner selection of iPhone4, Apple Company should take consideration of "green criteria" such as carbon emission, and choose environmental-friendly partners.

Figure 6.1 illustrates the model of PSP in iPhone4, the ovals denoted the parts in iPhone4, the box denoted the suppliers, and each supplier can provide one or several parts for the iPhone4. The problem is to choose the best supplier for each part to maximize the benefit and quality and minimize the risk and production time.

Figure 6.2 shows the PSP in general. A product consists of N parts. And there are M suppliers. Each supplier can provide one or several parts for the product. The PSP is to select suitable supplier for each part in the products. The selection of part supplier is decided by several criteria like cost, time, and carbon emission and so on. Some of these criteria are to be minimized, and the others are to be maximized.

In order to solve this problem, a model of the problem needs to be established.

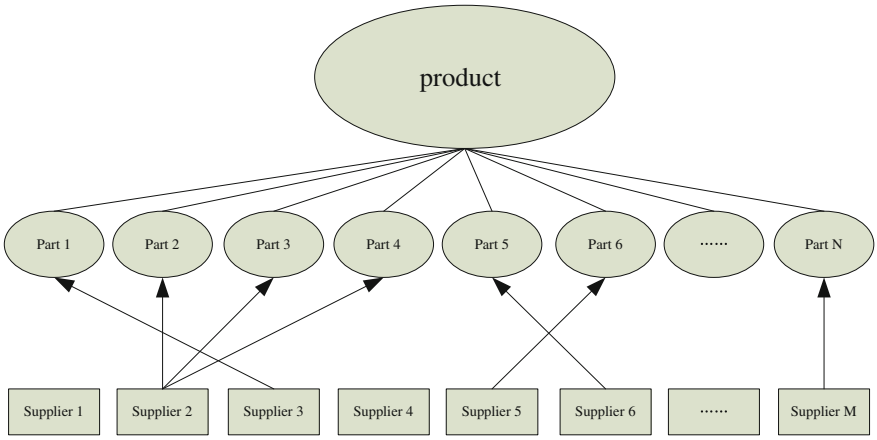


Fig. 6.2 Partner selection in VE

6.2.2 Formulation of the Partner Selection Problem (PSP)

It is assumed that a virtual enterprise has a project, p, to be completed, and p can be decomposed into M ($M = 1, 2, \ldots$) subprojects, i.e., $p = \{p_j | j = (1, 2, 3, \ldots, M)\}$, and the corresponding set of candidate partners for p_j is $x_j = \{x_{ji} | i = (1, 2, 3, \ldots, M_j)\}$, where $M_j \geq 1$ is the total number of candidate partners for p_j. Set $f_{jit} = 1$, if subproject p_j is contracted (or selected) to candidate partner x_{ji} for period t ($t = 1, 2, \ldots, T$); Otherwise $f_{jit} = 0$. Let ζ_{jit} denote the capacity (or available quantity of resources) of candidate partner x_{ji} provided for subproject p_j in period t.

Let the evaluation criteria of each partner be $A_j = \{a_n | n = (1, 2, 3, \ldots, N)\}$, and N is the number of criteria. In this chapter, four criteria such as *completion cost*, *risk*, quality, and *flexibility* represented by $c_{ost}, r_{isk}, q_{uality}, f_{lexibility}$ are considered. Let $c_{ost}(x_{jit})$ denote the *cost* for p_j executed on x_{ji}, and $r_{isk}(x_{ji}), q_{uality}(x_{ji})$, $f_{lexibility}(x_{ji})$ are the evaluation scores of criteria *risk*, *quality*, and *flexibility,* respectively. Therefore, one has $A = \{a_1, a_2, a_3, a_4\} = \{c_{ost}(x_{ji}), r_{isk}(x_{ji}), q_{uality}(x_{ji}), f_{lexibility}(x_{ji})\}$ and $N = 4$ in this article.

Let the beginning and completion times of p_j be tb_j and tc_j, respectively. Let the transporting cost between p_j (it is assumed that p_j is constructed to partner x_{ji}) and p_{j+1} (it is assumed that p_{j+1} is contracted to partner $x_{(j+1)k}$) be $tc_{ost}(x_{ji}, x_{(j+1)k})$. According to [37], a directed acrylic graph is set up to represent the precedence relationship among these subprojects. If p_j can only begin after the completion of p_k, and arc $(k, j) \in E$ is added to the graph, where $k, j = 1, 2, \ldots, M$ and E is the edge set representing the precedence relationship.

In this chapter, the aim of PSP is to select an optimal partner for each subproject and organize a dynamic alliance to completing project p while minimizing the total

cost and risk, and maximizing the quality and flexibility with the budget and deadline constraints. The partner selection problem can be formulated as follows:

Objective functions

$$\text{Min } C_{ost}(p) = \sum_{t}^{T}\sum_{j=1}^{M}\sum_{i=1}^{M_j} f_{jit}c_{ost}(x_{ji}) + \sum_{t}^{T}\sum_{j=1}^{M}\sum_{i=1}^{M_j}\sum_{k=1}^{M_{j+1}} f_{jit}f_{(j+1)kt}tc_{ost}(x_{ji}, x_{(j+1)k}) \tag{6.1}$$

$$\text{Min } R_{isk}(p) = \sum_{t}^{T}\sum_{j=1}^{M}\sum_{i=1}^{M_j} f_{jit}r_{isk}(x_{jit}) \tag{6.2}$$

$$\text{Max } Q_{uality}(p) = \sum_{t}^{T}\sum_{j=1}^{M}\sum_{i=1}^{M_j} f_{jit}q_{uality}(x_{jit}) \tag{6.3}$$

$$\text{Max } F_{lexibility}(p) = \sum_{t}^{T}\sum_{j=1}^{M}\sum_{i=1}^{M_j} f_{jit}f_{lexibility}(x_{jit}) \tag{6.4}$$

Subject to

$$\text{Max}(tc_1, tc_2, \ldots, tc_j, \ldots, tc_m) \le T_{Due} \tag{6.5}$$

$$\text{where } tc_j = tb_j + t_{x_{ji}} \text{ and } tb_j = \max(tc_k, \forall k : (k,j) \in E) \tag{6.6}$$

$$\sum_{t}^{T}\sum_{j=1}^{M}\sum_{i=1}^{M_j} f_{jit}c_{ost}(x_{ji}) + \sum_{t}^{T}\sum_{j=1}^{M}\sum_{i=1}^{M_j}\sum_{k=1}^{M_{j+1}} f_{jit}f_{(j+1)kt}tc_{ost}(x_{ji}, x_{(j+1)k}) \le B_{udget} \tag{6.7}$$

$$\sum_{t}^{tc_{x_{ji}}}\sum_{j=1}^{M} f_{jit}\zeta_{jit} \le Q_i \tag{6.8}$$

$$\sum_{t}^{tc_{x_{ji}}}\sum_{j=1}^{M} f_{jit} = 1 \tag{6.9}$$

Equation (6.5) states that the project must be completed no later than the project deadline T_{Due}. Equation (6.6) describes that the total cost (including the financial cost and transport cost) cannot be larger than the global budget B_{udget} for project p. Equation (6.7) imposes that, for any candidate partner i, can provide up to capability of Q_i in that period, and Constraint (8) states that for any period for a given subproject, only one candidate partner can be selected.

In the following section, the fitness function for the solutions to PSP is as follow:

$$\text{Min} \quad fitness(p) = O_{bjective}(p) + Constraint(p) \tag{6.10}$$

where

$$O_{bjective}(p) = \alpha_1 C_{ost}(p) + \alpha_2 R_{isk}(p) + \alpha_3 / Q_{uality}(p) + \alpha_4 / F_{lexibility}(p) \tag{6.11}$$

$$C_{onstraint}(p) = \begin{pmatrix} \beta_1 \times \max\left(0, \text{Max}(tc_1, tc_2, \ldots, tc_j \ldots, tc_m) - T_{Due}\right) \\ + \beta_2 \times \max(0, Cost(p) - B_{udeget}) \\ + \beta_3 \times \sum_{j=1}^{M} \sum_{i=1}^{M_j} \max(0, \sum_{t}^{tc_{xji}} \sum_{j=1}^{M} f_{jit} \zeta_{jit} - Q_i) \end{pmatrix} \tag{6.12}$$

In Eq. (6.10), $Objective(p)$ is the original minimization objective function of the partner selection problem. It combines the four optimization objective (cost minimization, risk minimization, quality maximization, and flexibility maximization) into a single function by four scaling factors, $\alpha_1, \alpha_2, \alpha_3, \alpha_4$. The factors $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are used as the corresponding weighted parameters to control the relative significance of each objective functions. They normalize the values of $C_{ost}()$, $R_{isk}()$, $Q_{uality}()$, and $Flexibility()$ to comparable ranges such that $Objective(p)$ will not be dominated by a single objective. $C_{onstraint}(p)$ is the penalty function to estimate the infeasibility of a solution. It is the quantified amount of mismatch if a solution is infeasible, otherwise $C_{onstraint}(p)$ is set to 0. In Eq. (6.11), the parameters $\beta_1, \beta_2, \beta_3$ are the weights controlling the relative significance constraints, respectively.

6.3 GA-BHTR for PSP

6.3.1 Review of Standard GA

The flowchart of a regular GA is shown in Fig. 6.3, and the brief workflows when using a regular GA to address an optimization problem are as follow [41]:

Step 1 A proper representation and fitness evaluation method is selected or designed.

Step 2 An initial population consisting of a number of individuals (or candidate solutions) is randomly generated.

Step 3 The quality or the fitness value of the specific optimization problem of each individual in the population is evaluated using the selected evaluation method.

Step 4 If the termination criterion is met, the best solution found thus far is returned.

Step 5 Some individuals are selected from the current population according to their fitness values. A new population is generated by applying the genetic operations such as reproduction, crossover, and mutation.

Step 6 Repeating Steps 3–5 until the stop criterion is satisfied.

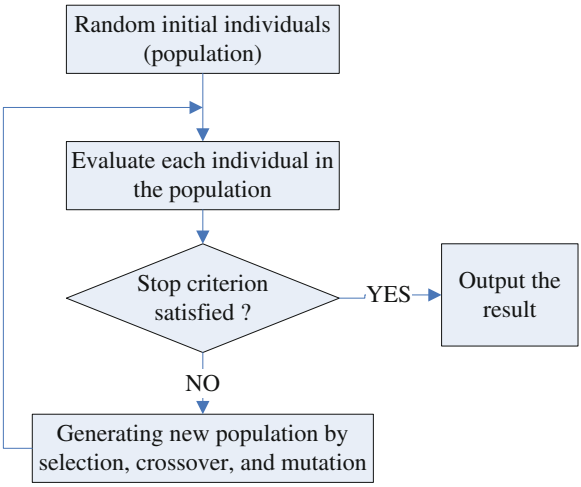


Fig. 6.3 Flowchart of a regular genetic algorithm

During the implementation of a regular GA, an iteration (i.e., repeating Steps 3–5) is called a generation. When using GA to address some complex engineering optimization problems, the key issues are individual representation, fitness evaluation, and the crossover and mutation operations.

6.3.2 Framewrok of GA-BHTR

In order to overcome the shortcomings of the standard GA mentioned in the introduction for addressing PSP, some new configurable components need to be added in the above framework. By means of the binary head and transitive reduction, the framework of GA-BHTR for PSP is proposed as shown in Fig. 6.4.

Compared to a regular GA, the red boxes are the new added improvement strategies and the blue boxes are the reconstruction of existing operators. With the red boxes, the original algorithm can be improved by configurable component. With the blue boxes, the structure of searching can also be adjusted flexibly. For solving specific PSP, the following problems need to be considered.

- How to establish the function of fitness for PSP in GA-BHTR?
- How to simplify the graph that represents the precedence relationship among the subprojects in PSP?
- How to distribute initial solutions to the different communities while maximizing the differences among these communities in order to enhance the diversity of GA-BHTR?
- How to avoid GA-BHTR from converging to a local best solution by using catastrophe?

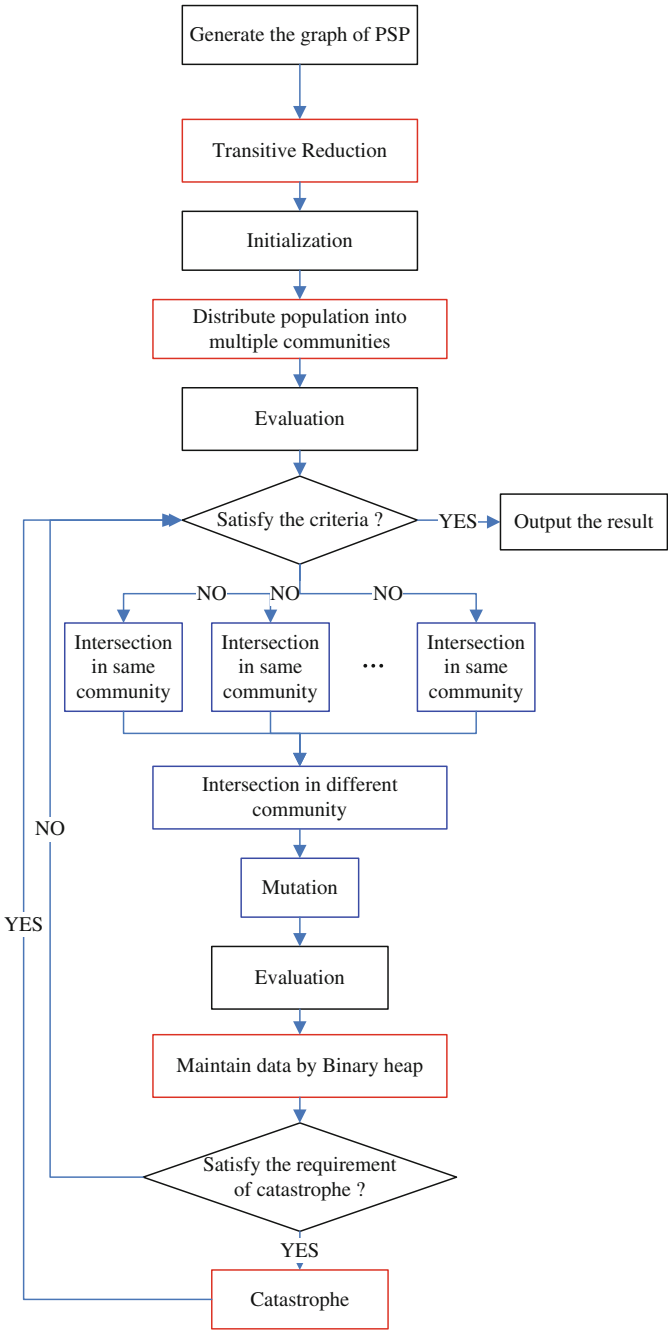


Fig. 6.4 The framework of the GA-BHTR for PSP

- How to maintain the solutions using the binary heap under the requirements of enhancing the diversity and reducing the computational time of GA-BHTR when implementing specific operations of intersection and mutation?

The following sections will describe the way the proposed GA-BHTR addresses the above problems in detail.

6.3.3 Graph Generation for Representing the Precedence Relationship Among PSP

When a specific PSP is given, the fist problem to be addressed is to establish the graph which represents the precedence relationship among the subtasks or sub-projects of a specific PSP. In this study, the topology of the graph is randomly generated. The related algorithms for generating the graph are as follows.

```
class tProjectSet{
private:
    bool inset[ProjectCount];
public:
    tProjectSet(){
        memset(inSet, false, sizeof(inSet));
    };
    void operator += (int x){
        inSet[x] = true;
    };
    void operator |= (const tProjectSet &x){
        For (int i = 0; i < ProjectCount; i ++) if (x.inSet[i]) inSet[i] = true;
    };
    Bool have(const int pro) const {return inSet[pro];};
    tProjectSet prevSet[ProjectCount];
    tProjectSet nextSet[ProjectCount];
    bool linkExisted[ProjectCount][ProjectCount];
    for (int i = 0; i < ProjectCount; i ++){
        prevSet[i] += i;
        nextSet[i] += i;
    }
    memset(linkExisted, false, sizeof(linkExisted));
    for (int i = 0; I < RelationCount; i ++){
        int p, n;
        do { p = randPro_1, n = randPro_2;} while (linkExisted[p][n] ||
nextSet[n].have(p));
        printf("%d %d\n", p, n);
        linkExisted[p][n] = true;
        for (int pp = 0; pp < ProjectCount; pp ++) if (prevSet[p].have(pp) {
            nextSet[pp] |= nextSet[n];
        }
        for (int nn = 0; nn < ProjectCount; nn ++) if (nextSet[n].have(nn)) {
            prevSet[nn] |= prevSet[p];
        }
    }
}
```

Note:

- P and N denote two random points (i.e., subtasks or subprojects);
- LinkExisted[p][n] is designed to check whether there is an edge between p and n;
- NextSet[n].have(p) is designed to check whether the point p is included in the subsequence point set of n;
- PrevSet[p].have(pp) is designed to check whether pp is included in the pre-projects set of p.

The fitness function is used to evaluate the quality of each candidate solution. When a solution is selected, the fitness value is the total cost of all the selected partners in the solution while $\text{Max}\{tc_j\} \leq T_{Due}$. The fitness of a solution is obtained using the following two steps in this study.

First, the new order of the projects needed is established according to their topology order. The topology order can be obtained by repeatedly finding projects without any pre-projects, and the corresponding algorithms are as follows:

```
void dfs(int p){
    vist[p] = true;
    for (int i = 1; i <= project_relation[p][0]; i ++){
        int next = project_relation[p][i];
        if (!vist[next]) dfs(next);
    }
    Toplist[--listp] = p;
}

void topsort(){
    listp = project_num;
    memset(vist, false, sizeof(vist));
    for (int i = 0; i < project_num; i ++)
        if (in_degree[i] == 0) dfs(i);
}
```

Note:

- The array vist is a sign to demonstrate whether the project has been selected;
- The array toplist is used to store the final topology order of the graph;
- The array project_relation stores the relation among projects.

Second, a simple dynamic programming is used to obtain the value of $\text{Max}\{tc_j\}$, which is the critical path of the graph. Since the topology order of the problem has Markov property, hence if $\text{Max}\{tc_j\} \leq T_{Due}$, the solution is valid. Meanwhile, the value of fitness is also calculated by $f(x) = \sum_{j=1}^{m} c_{x_{jk}}$. The corresponding algorithms of the dynamic programming for obtaining the value of $\text{Max}\{tc_j\}$ are as follows.

```
memset(startTime, 0, sizeof(startTime));
for (int i = 0; i < prog->projectCount; i ++){
    int s = prog->topologyOrder[i];
    endTime[s] = startTime[s] + pro_par(s).time;
    if (endTime[s] > maxEndTime){
        maxEndTime = endTime[s];
        if (maxEndTime > prog->timeLimit) break;
    }
    for (int I = prog->nextCount[s] – 1; i >= 0; i --){
        int t = prog->next[s][i];
        if (endTime[s] > startTime[t]) startTime[t] = endTime[s];
    }
}
if (maxEndTime > prog->timeLimit)
    fitness = TOO_MUCH_COST;
else{
    fitness = 0;
    for (int i = prog->projectount – 1; i >= 0; i --) totalCost += pro_par(i).cost;
}
```

Note:

- topologyOrder[] is used to save the sorted topology order;
- endTime[s] denotes the earliest end time of s;
- startTime[s] denotes the earliest beginning time of s while its pre-objects are all finished;
- pro_par(s).time is the implementing time for the selected partner to execute task s;
- pro_par(s).cost is the cost for the selected partner to execute task s;
- fitness is the fitness value of a solution.

The time complexity for obtaining the topology order is $O(E)$, where E is the number of relation, and the time complexity of the dynamic programming is also $O(E)$. Therefore, the time complexity for calculating the fitness of the solutions is $O(E)$.

Although the fitness value can be calculated using the $O(E)$ algorithm, which is a relatively fast method, there is also much unnecessary calculation. The graph for the precedence relationship of PSP is a directed acyclic graph. In this graph, some of the topological relation is unnecessary because such relations can be totally replaced by some other edges. Moreover, these edges will never be used as the critical path. For example, in Fig. 6.5, the relationship between p_i and p_j is unnecessary when the critical path is $p_i \rightarrow p_k \rightarrow p_j$. The reasons are follows.

Let the earliest time for completing task i, j, and k be $endTime[i]$, $endTime[j]$, and $endTime[k]$, and the processing time of task i, j, and k be $TimeCost[i]$, $TimeCost[j]$, and $TimeCost[k]$, respectively. Then the maximal executing time for selecting the path $p_i \rightarrow p_k \rightarrow p_j$, denoted as $Time_{i\rightarrow k\rightarrow j}$, is as follow.

$$Time_{i\rightarrow k\rightarrow j} = endTime[j] = \max(endTime[i], endTime[k]) + TimeCost[j] \quad (6.13)$$

And the maximal executing time for selecting the path $p_i \rightarrow p_j$, denoted as $\text{Time}_{i\rightarrow j}$, is as follow:

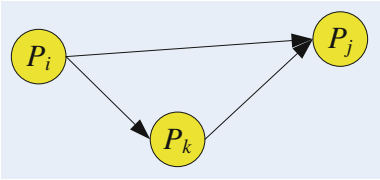


Fig. 6.5 Precedence relationship between p_i, p_j and p_k

$$Time_{i\to j} = endTime[j] = endTime[i] + TimeCost[j] \tag{6.14}$$

Because $endTime[k] = endTime[i] + TimeCost[k]$, and $endTime[i] > 0$, $Max(endTime[i], endTime[k]) = endTime[k] > endTime[i]$. Therefore $Time_{i\to k\to j} > Time_{i\to j}$, i.e. the executing time for path $p_i \to p_k \to p_j$ is larger than the path $p_i \to p_j$. It means that when obtaining the value of $\text{Max}\{tc_i\}$, the relationship $p_i \to p_j$ is unnecessary and can be reduced in this example.

Therefore the edge between p_i and p_j can be reduced so as to reduce the time complexity for addressing PSP. For projects p_i, p_j and p_k, if there are existing direct or indirect relationship between p_i and p_k, and p_k and p_j, while p_i and p_j have direct relationship, then the direct relationship between p_i and p_j can be reduced. The related reducing algorithms are as follows.

```
for (int k = 0; k < projectCount; k ++) //enumerate projects
    for (int i = 0; i < projectCount; i ++)
        if (inderectLink[p][i] || link[i][k]) //exist relation
            for (int j = 0; j < projectCount; j ++)
                if (indirectLink[k][j] || link[k][j]) //exist relation
                    indirectLink[i][j] = true; //calculate indirect relation
for (int i = 0; i < projectCount; i ++) //enumerate projects
    for (int j = 0; j < projectCount; j ++)
        if (indirectLink[i][j] && link[i][j])
            Link[i][j] = false; //delete the unnecessary relation
```

Note:

- The array Link represents the direct relations in the graph;
- The array indirectLink represents the indirect relations in the graph;
- The operation that setting Link[i][j] to value of false means deleting the edge between them.

After the operation of the above transitive reduction, the graph can be simplified greatly and the time complexity for calculating the fitness is reduced significantly.

6.3.4 Distribute Individuals into Multiple Communities

In the traditional GA, there is usually only one community (or populations) during evolution, which results in most of solutions converge to a constant value after a number of generations. One of the reasons is that, during the evolution, the solutions (or individuals) of the next generations are created towards the best solutions produced in the last generation. It keeps in view the solutions similar to the best solution in the community in the next generation after a number of generations. It results in the early convergence of a solution to a local best solution.

Therefore, some researchers have used multiple communities to avoid the solutions converge to a constant value too early during the evolution [42–44]. Since the individuals in different communities are independent and different, some mechanisms have been designed for the intersection operation between the individuals among different communities, therefore the diversity of solutions is enhanced, and the probability that the algorithms may converge too early during evolution can be reduced significantly. For example, if there are two individual G_1 in community C_1 and individual G_2 in community C_2, where G_1 has a good segment S_1 and G_2 has a good segment S_2. S_1 and S_2 may overlap in position. If an intersection operation is conducted between these two individuals in C_1 and C_2, then they can "learn" from each other and generate a better segment. However, if there is only one community, all the individuals may evolve to a similar solution. The processes using multiple communities during the evolution are shown in Fig. 6.6.

In Fig. 6.6, the *i*th community and *j*th community reproduce in every continuous generation. After a certain number of generations, they are selected randomly for intersection. When using multiple communities in GA, the distribution of initial solutions to the different communities is very important. If the difference of the solutions in different communities is too small, then there is no distinct advantage for using multiple communities compared to using a single community. In order to release the biggest advantage using multiple communities, the

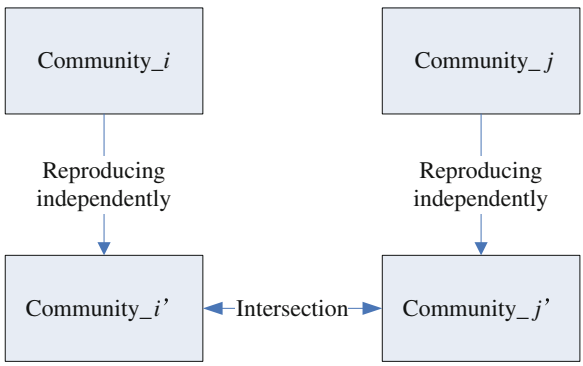


Fig. 6.6 Illustration of intersection between multiple communities

difference among different communities should be maximized when distributing the initial solutions to the different communities.

However, the initial solutions are generated randomly when using multi communities in most related works. The distribution of initial solutions into multiple communities (DISMC), while maximizing the difference among different communities, can be considered as an NPC problem. In the following sections, a regular GA is employed to solve the DISMC problem.

It is assumed that there are $n(n = 1, 2, 3\ldots)$ initial individuals marked with $G_1, G_2, \ldots, G_n$, then the DISMC problem is to distribute the n initial individuals into $N_c(N_c = 1, 2, 3\ldots)$ communities while maximizing the difference among each two pairs of communities.

Let d_{ij} be the value of the difference between two individuals $G_i = \{x_{1i_1}, x_{2i_2}, \ldots, x_{ki_k}, \ldots, x_{mi_m}\}$ and $G_j = \{x_{1j_1}, x_{2j_2}, \ldots, x_{kj_k}, \ldots, x_{mj_m}\}$, then d_{ij} is calculated as follows:

$$d_{ij} = \begin{cases} 0 & (i = j) \\ \sum_{k=1}^{m} S_{ijk} & (i \neq j, \text{and if } x_{k,k_i} = x_{k,k_i}, \text{then } S_{ijk} = 0, \text{else } S_{ijk} = 1) \end{cases} \tag{6.15}$$

The difference information matrix $\mathbf{D_{iff}}$ of all the individuals can be represented as follows.

$$\mathbf{D_{iff}} = \begin{bmatrix} d_{11} & \cdots & d_{1j} & \cdots & d_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ d_{i1} & \cdots & d_{ij} & \cdots & d_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ d_{n1} & \cdots & d_{nj} & \cdots & d_{nn} \end{bmatrix} \tag{6.16}$$

Let $num_a(num_a = 1, 2, 3\cdots)$ and $num_b(num_b = 1, 2, 3, \cdots)$ be the numbers of the individuals in the ath and bth communities, then the difference, D_{ab}, between the ath community and bth community is defined as follows:

$$D_{ab} = \sum_{i=1}^{num_a} \sum_{j=1}^{num_b} d_{ij} \tag{6.17}$$

Therefore, the best solution to the DISMC problem is to make the sum of the differences among each two pairs of communities, *SumDiff*, have the biggest value, i.e., maximize Eq. (6.18).

$$SumDiff = \sum_{a=1}^{N_c} \sum_{b=a+1}^{N_c} D_{ab} \tag{6.18}$$

The DISMC problem cannot be addressed using a simple Polynomial-level algorithm. In this chapter, a regular GA is employed for addressing the DISCM problem, and the corresponding steps are as follows:

Step 1 Randomly generate $m \times n$ $(n, m = 1, 2, 3\ldots)$ individuals, denoted as $G_1, G_2, \ldots, G_n \ldots, G_{2n}, \ldots, G_{mn}$, as the initial community to be distributed, and divide them into *m*groups evenly and store them in m arrays, i.e., $Arr_1, Arr_2, \ldots, Arr_i, \ldots, Arr_m$, and $Arr_i = [G_{(i-1)n} G_{(i-1)n} \ldots G_{in}]$.

Step 2 Divide the n individuals in each array, e.g., Arr_i, into $N_c (N_c = 1, 2, 3\ldots)$ communities orderly, i.e., assign the first NUM_1 individuals into the first community, and the following NUM_2 individuals into the second community, the following NUM_j individuals into the jth $(j = 1, 2, 3\ldots N_c)$ community. Obviously, after this execution, a simple solution for the DISMC problem can be generated.

Step 3 Using the following three operations to generate new offsprings for the specific DISMC problem while keeping some of the characteristics of the former solution.

(a) Reverse one segment in the individual. For example, randomly take two numbers i and j, reverse the segment from G_i to G_j, as shown in Fig. 6.7.

(b) Exchange two random segments of Arr_i. For example, if one randomly obtains 3 numbers k, l_1 and l_2 (k, l_1 and l_2 are all positive integers, and $k - l_1 \geq 1$ and $k + l_2 \leq n$), then exchange the related segments as shown in Fig. 6.8.

(c) Use the mutation operation to keep the diversity of the community. For each individual, randomly choose two positions i and j, then exchange the value of them, as shown in Fig. 6.9.

Step 4 In order to keep the volume of community at a constant value and enhance the efficiency, some individuals would need to be deleted according to their fitness value. The larger fitness value is, the higher the survival probability of the individual will be.

Step 5 Evaluate the *SumDiff* of the new solutions for a specific DISMC problem.

Step 6 Repeat Steps 3–5 until a relatively satisfactory solution has been found.

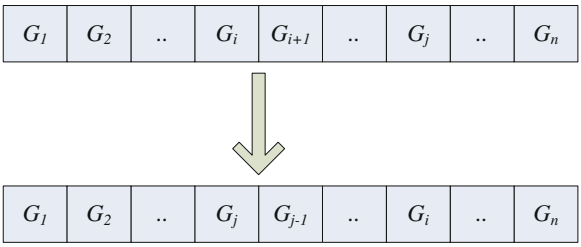


Fig. 6.7 Illustration of the reverse operation

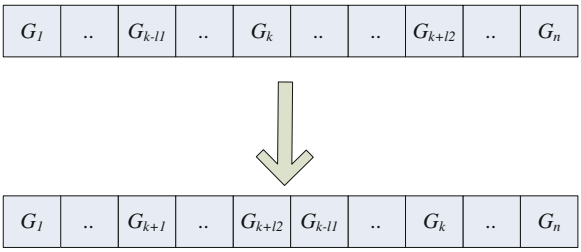


Fig. 6.8 Illustration of the exchange operation

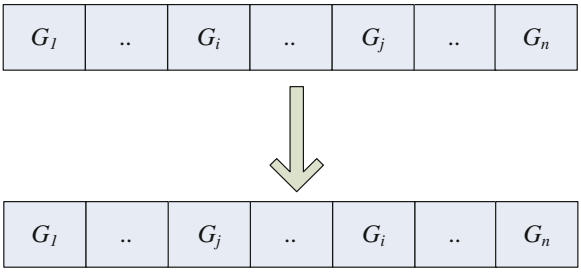


Fig. 6.9 Illustration of the mutation operation

6.3.5 Intersection and Mutation in GA-BHTR

Similar to the regular GA, the operations of intersection and mutation are also used in GA-BHTR. For a specific PSP, the sequence for a candidate solution is not permutation, therefore an individual in GA-BHTR cannot be reproduced by simply exchanging its two segments, and as such an exchange is meaningless because it does not keep any information of the former individuals. Therefore, some specific intersection and mutation operations are used in GA-BHTR. In the implementation of GA-BHTR, the method of Russian roulette is used to select candidate solutions for intersection and mutation.

(a) Intersection in the same community
In GA-BHTR, the capacity of each community is maintained at a constant integer value N. Two candidate individuals, e.g., $G_i = \{x_{1i_1}, x_{2i_2}, \ldots, x_{ki_k}, \ldots, x_{mi_m}\}$ and $G_j = \{x_{1j_1}, x_{2j_2}, \ldots, x_{kj_k}, \ldots, x_{mj_m}\}$, are selected using the turntable method and a position k is selected randomly. Then the two individuals are cut at position k, and their information after the position k is exchanged. It means that the two individuals change from $x_{1i_1}x_{2i_2}\ldots x_{ki_k}\ldots x_{ni_n}$ and $x_{1j_1}x_{2j_2}\ldots x_{kj_k}\ldots x_{nj_n}$ to $x_{1i_1}x_{2i_2}\ldots x_{kj_k}\ldots x_{nj_n}$ and $x_{1j_1}x_{2j_2}\ldots x_{ki_k}\ldots x_{ni_n}$.The corresponding process is illustrated in Fig. 6.10. For obtaining additional characteristics of the individuals, the intersections are conducted several

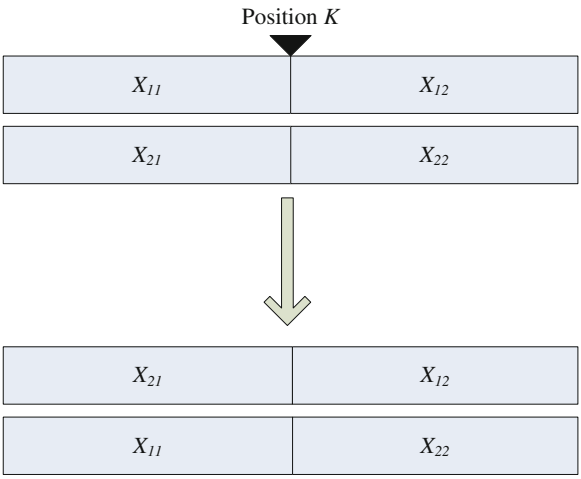


Fig. 6.10 Illustration of an intersection in the same community

times. As a result, V ($V = 1, 2, 3 \cdots$ and it is much bigger than N which is the constant capacity of community) candidates for each community are generated.

For convenience, x_{11}, x_{12}, x_{21} and x_{22} are used to represent the segment divided from the *kth* position.

(b) Intersection between different communities

After an amount of generations during the evolution of the algorithm, some good individuals are generated in the different communities, and the information from the different communities can be exchanged in order to enhance the performance of the algorithm. First, two communities are selected, and two individuals are selected from the two respective communities. The probability of the individuals being chosen was calculated using the same method as described above, while the communities are selected based on equal probability. The detailed algorithms for the intersection are the same with that in the same community.

(c) Mutation

Sometimes, in order to move out from the current local optimal or to enhance the exploration ability of the algorithm and search in an even larger searching space, the mutation operation is usually used. In GA-BHTR, the flow of mutation is realized as follows: first an integer number k ($k = 1, 2, 3 \ldots$ and $k < m$) is selected from an individual, then the number in the kth position is changed into a random x, where x is less than the maximal number of its partner as shown in Fig. 6.11.

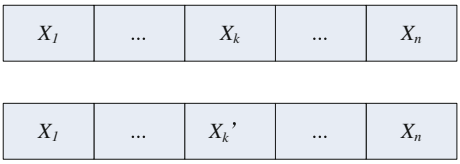


Fig. 6.11 Illustration of mutation

6.3.6 Maintain Data Using the Binary Heap

When some generations have completed reproducing, there will be a total of V new individuals in each community as mentioned before. Obviously, a larger V can enhance the diversity of the algorithm. However, V is much larger than N which is the capacity of community, and a larger V will make the algorithm more time-consuming. Therefore, some solutions with poorer quality will have little chance to reach the best solution and will need to be killed due to capacity limitation and the consideration for better efficiency.

Some researchers have used the naïve method to kill the redundant solutions. First they save the value of the probability of the elements in V to be deleted in an array, then obtain a random number between 0 and $totalRate - 1$ ($totalRate$ is the sum of the probabilities of all elements in V to be killed), and use the sequential search method to find the elements to be deleted. The time complexity for the above operation is $O(N_{kill}{}^{*}V)$, and N_{kill} ($N_{kill} = 1, 2, 3\ldots$) is the number of the individuals to be killed. The detailed corresponding algorithms are as follows:

```
int rate[TURNTABLE_SIZE], totalRate;
void int (const int *rateList, const int size){
        totalRate = 0;
        for (int i = size - 1; i >= 0; i --)
                totalRate += (rate[i] = max(rateList[i], 0)) ;
}
int choose(){
        if (totalRate == 0) return -1;
        int rp = rand() % totalRate, r = 0;
        while (rp >= rate[r]) rp -= rate[r++];
        totalRate -= rate[r];
        rate[r] = 0;
        return r;
}
```

Note: The array rate stores the probability of every element being killed.

However, when V is very large, the above naïve algorithm becomes very time-consuming. Therefore, in this chapter, the authors have designed a binary heap to maintain the data, and use the turntable maintained by the binary heap to kill the redundant solutions in V, which will reduce the time complexity to $O(N_{kill}{}^{*} \log V)$. The simple algorithms for constructing the binary heap are as follows.

```
for (int i = size; i > 0; i ++)
        totalRate[i] = max(rateList[i - 1], 0) + totalRate[i * 2] + totalRate[i * 2 + 1];
```

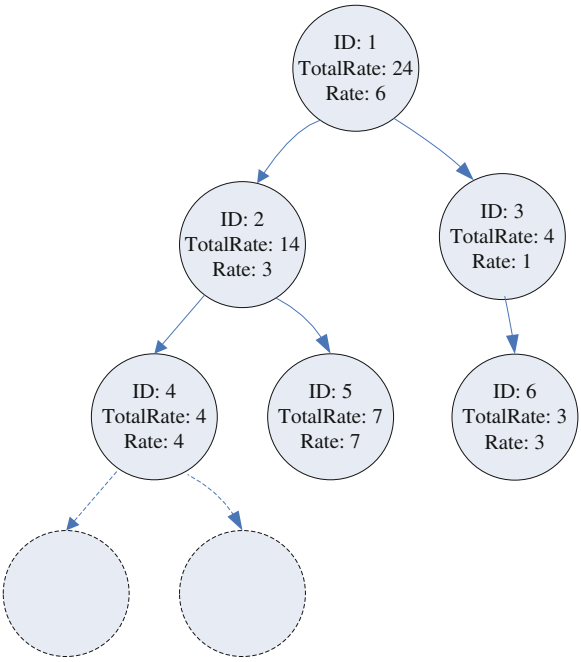


Fig. 6.12 The graphical display of a binary heap

Note:

Variable I denotes the uniquely ID of each node (i.e. an element in *V*);

rateList is used to store the probability of each element to be killed, and its value is calculated according to its fitness value;

totalRate is the sum of the probabilities of the element and its total sub-elements to be killed.

In the binary heap, each node manages a section of data and has two sub-nodes. The graphical display of the binary heap is shown in Fig. 6.12.

In Fig. 6.12, 'ID' is the uniquely identification of an element, i.e., the solution, in *V*; the 'Rate' is the probability of an element to be killed, and its value is calculated according to its fitness value; the 'TotalRate' is the sum of the probabilities of the element and its total sub-elements to be killed.

Moreover, in Fig. 6.12, one can see every node in the binary heap represents an element in the turntable. When searching a node in the binary heap, one can first check whether the node is located at the left section of the current node. If the answer is yes, then look for the node in the left section. Otherwise, one should check whether the node is located at the right section of the current node. If the answer is yes, then one should look for the node in the right section. If both the answers are no, then the current node is the node one is looking for. For example, if one wishes to search and delete the solution with a random number 12 in the binary heap as shown in Fig. 6.12, the process of searching is as follows:

Step 1 Check the *totalRate* value of the left sub-tree (node 2) of node 1. Its *totalRate*(14) is bigger than the number(12), then continue to search in its left sub-tree.

Step 2 Check the *totalRate* value of the left sub-tree (node 4) of node 2. Its *totalRate*(4) is smaller than the number(12), so let the number(12) minus the *totalRate*(4) of node 4. Then its current value is 8.

Step 3 Check the *rate* value of node 2. One can see its *rate*(3) is smaller than the number (8), so minus the value by the *rate* of node 2(3) and continue to search in its right sub-tree. This time its current value is 5.

Step 4 Check the value of the left sub-tree of node 5 and find that it does not have a left sub-tree. Then one finds the *rate*(7) of this node 5 is bigger than the number(5), so the number belongs to this node.

Step 5 For avoiding the numbers from appearing on the same node, after finding one node, one would need to make its value of *rate* to zero and modify the *totalRate* of its father node recursively.

One can see the time complexity of the above process is from the depth of the binary heap. Moreover, the structure of the binary heap will never be changed by the operation. The corresponding algorithms are as follows.

```
int totalRate[TURNTABLE_SIZE * 2 + 2];
void init(const int *rateList, const int size){
    memset(totalRate, 0, sizeof(totalRate));
    for (int i = size; i > 0; i --)
        totalRate[i] = max(rateList[i − 1], 0) + totalRate[i * 2] + totalRate[i * 2 + 1];
}
int choose(){
    if (totalRate[1] == 0) return -1;
    int rp = mrand(0, totalRate[1] − 1), r = 1;
    while (rp < totalRate[r * 2] || rp >= totalRate[r] − totalRate[r * 2 + 1])
        r = (rp < totalRate[r * 2] ? r * 2: (rp -= totalRate[r] − totalRate[r * 2 + 1], r * 2 + 1));
    int clearRate = totalRate[r] − totalRate[r * 2] − totalRate[r * 2 + 1];
    for (int p = r; p > 0; p /= 2) totalRate[p] -= clearRate;
    return r − 1;
}
```

Note: The function **mrand(0, totalRate[1]-1)** is generate a random number between 0 and totalRate[1]-1.

This method allows the enlargement of the capacity of the community because one can maintain the data quickly by using the binary heap. Obviously, the larger the volume of data, the greater the advantage of the algorithm is.

6.3.7 The Catastrophe Operation

The solutions may still converge to a local optimal even after some mutation operations in the traditional GA. Under this condition, the algorithm should move out from the current local optimal, and search in an even larger searching space.

A lager mutation operator will be needed so as to enhance the exploration and find new search fields. In the proposed GA-BHTR, a giant mutation called the "catastrophe" is used to overcome this problem.

In the proposed GA-BHTR, $T^0(T^0 = 1, 2, 3 \cdots)$ is defined as the starting countdown number of the catastrophe at the beginning. If the fitness value of the best solution in the current generation does not increase, the countdown number will be minus one in a single generation. If the countdown number reaches zero, then the catastrophe happens. In this process of catastrophe, every individual in the community will mutate consecutively for many times. On the opposite, if the fitness value of the best solution in a generation increases, the starting value of the countdown number, T, for the next generation will be recalculated dynamically as follows [45]:

$$T = \max\left((T_{cata} - T'_{cata}) * \varphi, T_{cata}\right) \tag{6.19}$$

where T_{cata} ($T_{cata} = 1, 2, 3 \cdots$ and $T_{cata} \leq T^0$) is the last starting countdown value, T'_{cata} ($T'_{cata} = 1, 2, 3 \cdots$ and $T'_{cata} \leq T^0$) is the countdown value in the current generation, and φ is a constant coefficient, in the proposed GA-BHTR algorithm.

The related algorithms for the catastrophe operation are as follows:

```
void catastrophe_reset(){
    catastropheCountDown = (catastropheInit = max((int)((catastropheInit –
    catastropheCountDown * NEW_CATASTROPHE_COUNT_DOWN_RATE),
    catastropheInit));
}
void check_evolution(){
    if (check_best()){
        evolutionTimes ++;
        catastrophe_reset();
    }
    else{
        catastropheCountDown --;
        if (catastropheCountDown <= 0){
            init(indi[0], RENASCENCE_VARY_TIMES);
            catastropheTimes ++;
        }
    }
}
```

6.4 Simulation and Experiment

In order to test the effectiveness and efficiency of the proposed operations and algorithms in GA-BHTR for addressing the different sizes of the partner selection problems in a virtual enterprise, a group of experiments are conduced. All the codes of GA-BHTR are programmed using C++, and the algorithms are implemented on an Intel 1.83 MHz PC with 1 GB of RAM under Linux, Ubuntu 9.10.

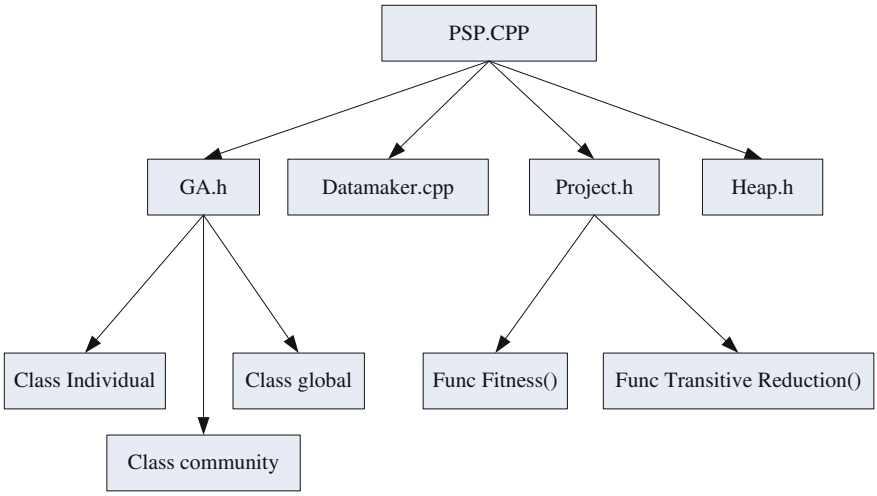


Fig. 6.13 Structure of the GA-BHTR

The structure of the GA-BHTR is shown in Fig. 6.13. The related key parameters of GA-BHTR are follows:

- The number of individuals is 100.
- The number of communities is 5.
- The volume of individuals generated in a single round is 1,900.
- The probability for mutation is 80 %.
- The probability for intersection between different communities is 10 %.
- The initial countdown value for applying the catastrophe operation is 800, i.e., $T^0 = 800$ in Eq. (6.19).
- $\varphi = 2.6$ in Eq. (6.19).

Each result is an average of ten test cases. The summary of the results for each operation in GA-BHTR is discussed in the following sections.

6.4.1 Effectiveness of the Proposed Transitive Reduction Algorithm

In this section, the time complexity for fitness evaluation is selected as the index to test the effectiveness of the proposed transitive reduction algorithm in GA-BHTR for addressing PSPs. The results are shown in Table 6.2.

In Table 6.2, for the number '20 ^ 20' in the column of 'solution space', the first two numbers '20' denote the number of the subprojects of the tested PSPs, while the next two numbers '20' denote the number of the candidate partners for each subproject. For example, 20 ^ 20 means there are 20 subprojects of the tested

Table 6.2 Efficiency of using transitive reduction operation in GA-BHTR

Solution space	Before using transitive reduction operation		After using transitive reduction operation	
	The number of relationship for the PSP (E)	Time complexity for fitness evaluation	The number of relationship for the PSP (E)	Time complexity for fitness evaluation
10 ^ 10	30	O(30)	10	O(10)
20 ^ 20	50	O(50)	20	O(20)
30 ^ 30	100	O(100)	38	O(38)
40 ^ 40	200	O(200)	45	O(45)
50 ^ 50	250	O(250)	64	O(64)

PSP and each subproject has 20 candidate partners for selection. The numbers in the second and the fourth columns in Table 6.2 are the number of relationships needs to be calculated before and after using transitive reduction algorithm, respectively, while the third and fifth columns are the corresponding time complexity for fitness evaluation.

It can be seen from Table 6.2 that, when using the proposed transitive reduction, the time complexity for fitness evaluation can be reduced greatly. The effectiveness of the proposed transitive reduction algorithm in GA-BHTR for addressing PSPs is quite apparent.

As many engineering optimization problems can be modeled and represented using a directed acrylic graph, therefore, the proposed transitive reduction algorithm can be used for reducing the computing complexity for other engineering optimization problems in addition to PSP.

6.4.2 Effectiveness of Multiple Communities

The aim of this group of experiments is to test the effectiveness for applying the strategy of multiple communities in GA-BHTR to address the different scales of the PSPs. The experimental results are shown in Table 6.3.

In Table 6.3, the meaning of the numbers in the solution space is the same with that in Table 6.2. The numbers in the second column denote the total evolution generations times for the corresponding algorithms in the experiments. The numbers in the third and fourth columns are the average minimal fitness value of the best solutions obtained from the algorithms for the PSPs tested under different solution space, and the values are the average of ten test cases.

It can be concluded from Table 6.3 that the effectiveness of multiple communities (here there are five communities) is not visible when the generation and solution space are relatively small, and some slight advantages can be achieved

Table 6.3 Performance comparisons between single community and multiple communities

Solution space	Number of generations	Average minimal fitness value using single community	Average minimal fitness value using five Communities
10 ^ 10	100	90.45	89.34
20 ^ 20	500	125.69	124.44
30 ^ 30	1,000	136.39	133.62
40 ^ 40	10,000	156.49	151.83
50 ^ 50	50,000	214.44	208.86

Table 6.4 Performance comparisons between with and without the consideration of DISMC problem

Solution space	Number of generations	Average minimal fitness value when considering the DISMC problem	Average minimal fitness value without consideration of the DISMC problem
10 ^ 10	100	87.53	89.34
20 ^ 20	500	120.74	124.44
30 ^ 30	1,000	129.53	133.62
40 ^ 40	10,000	146.72	151.83
50 ^ 50	50,000	203.60	208.86

when the generation and solution spaces become larger. But a better result can be obtained by applying some specific operations or algorithms to keep the diversity of the communities as discussed before.

6.4.3 Effectiveness of Multiple Communities While Considering the DISMC Problem

When using multiple communities in GA, the distribution of initial solutions to the different communities is very important. If the difference of the solutions in the different communities is too small, then there is no visible advantage for using multiple communities than using only one community, as shown in Table 6.3. In order to unleash the major advantages by using multiple communities and achieve better results and, in the second group experiments, the DISMC problems are considered, i.e., the difference among the different communities is maximized when distributing the initial individuals to the multiple communities using the method mentioned before.

The experimental results are shown in Table 6.4. It can be seen from Table 6.4 that the effectiveness of multiple communities while considering the DISMC problem is visible this time.

Therefore, the idea of maximizing the difference among different communities when using multi-communities or multi-populations in an evolutionary algorithm is very effective, especially for solving large scale and large solution space engineering optimization problems. Furthermore, it is of good reference value when designing parallel optimization algorithms running on high performance computers and multi-core CPUs in future.

6.4.4 *Effectiveness of the Catastrophe Operation*

The third group of experiments is designed to test the effectiveness for applying the catastrophe in GA-BHTR to address the different scales of the PSPs. The experimental results are shown in Table 6.5. It can be concluded from Table 6.5 that there is almost no advantage using the catastrophe when the generation is equal or less than 1,000. The reason is that the initial countdown value (i.e., the generation) for applying the catastrophe operation is 800. When the generation is equal to or less than 1,000, the catastrophe operation was not used or only used one or two times, hence the effectiveness of the catastrophe operation is negligible. However, when the generations and solutions space become larger, some visible advantages can be achieved as shown in Table 6.5.

6.4.5 *Efficiency of Using the Binary Heap*

The aim of the fourth group of experiments is to compare the efficiency, i.e., the total executing time, of the two methods, i.e., using binary head to maintain the individuals (i.e., the solutions) and without binary heap maintenance in GA-BHTR. The experimental results are shown in Fig. 6.14. The horizontal coordinate in Fig. 6.14 denotes the numbers of evolution generations and the size of the solution space of the tested PSPs, while the values in the vertical coordinate are the total executing time for the two methods under different conditions.

Table 6.5 Performance comparisons between with and without the catastrophe operation

Solution space	Number of generations	Average minimal fitness value with the catastrophe operation	Average minimal fitness value without the catastrophe operation
10 ^ 10	100	86.34	87.53
20 ^ 20	500	118.51	120.74
30 ^ 30	1,000	126.91	129.53
40 ^ 40	10,000	143.92	146.72
50 ^ 50	50,000	200.13	203.60

It can be seen from Fig. 6.14 that, by using the proposed binary heap method to maintain the solutions, the total executing time for GA-BHTR to address some PSPs is shorter than the method without binary heap maintenance as shown in Fig. 6.14a. The comparative advantage, however, is not significant. One of the reasons is due to the small number of the executed generations, and the overall time required for the experiments in Fig. 6.14a is comparatively lower. As a result, although using the binary heap to maintain the data can save some executing time, the saving is quite limited. The other reason is due to the small value of the first point in the curve in Fig. 6.14a which is 0.31 from the proposed algorithm, which is much smaller than the value of the last point (539.11) due to the incensement of the solution space and the generation number. Therefore, the efficiency of the proposed method is not outstanding compared with traditional method as shown in Fig. 6.14a.

With the increase of the solution space and the number of generations of the tested PSPs, the efficiency using the binary heap method to maintain data in GA-BHTR is higher, as shown in Fig. 6.14b. Hence, the proposed method is more suitable for addressing complex engineering optimization problems with large scale solution space.

In Fig. 6.14, the symbol 'Num_Gen' denotes the number of generations and 'SP' denotes the solution space, respectively.

Although the method using the binary heap to maintain the data can enhance the efficiency of GA-BHTR, the effect is not very distinct as demonstrated in Fig. 6.14a. In order to test how much the executing time can be saved using the proposed algorithm when killing the redundant solutions with poorer quality from V, an additional group experiments are conducted to compare the efficiency between the binary heap method and naïve algorithm. The experimental results are shown in Table 6.6 and Fig. 6.15. The numbers of the first column in Table 6.6 are the size of the turntable used both in the two methods. The values in the second and third columns are the implementing time for the two methods to kill the redundant solutions with poorer quality from V. Each result is an average of ten test cases, and in each case the points are selected in the turntable maintained using binary heap for 10,000 times.

From the results shown in Table 6.6 and Fig. 6.15, one can conclude that by using the binary heap to maintain the data, the time for killing redundant solutions with poorer quality from V is less than using naïve algorithm. The efficiency using the binary heap to maintain the data is more distinct under this group experiments.

As the proposed method using the binary heap to maintain the redundant solutions is used as an independent configurable component to maintain the data during evolution in the proposed GA-BHTR, it can also be used in other optimization algorithms, such as particle swarm optimization (PSO) and ant colony optimization (ACO), for enhancing their efficiency.

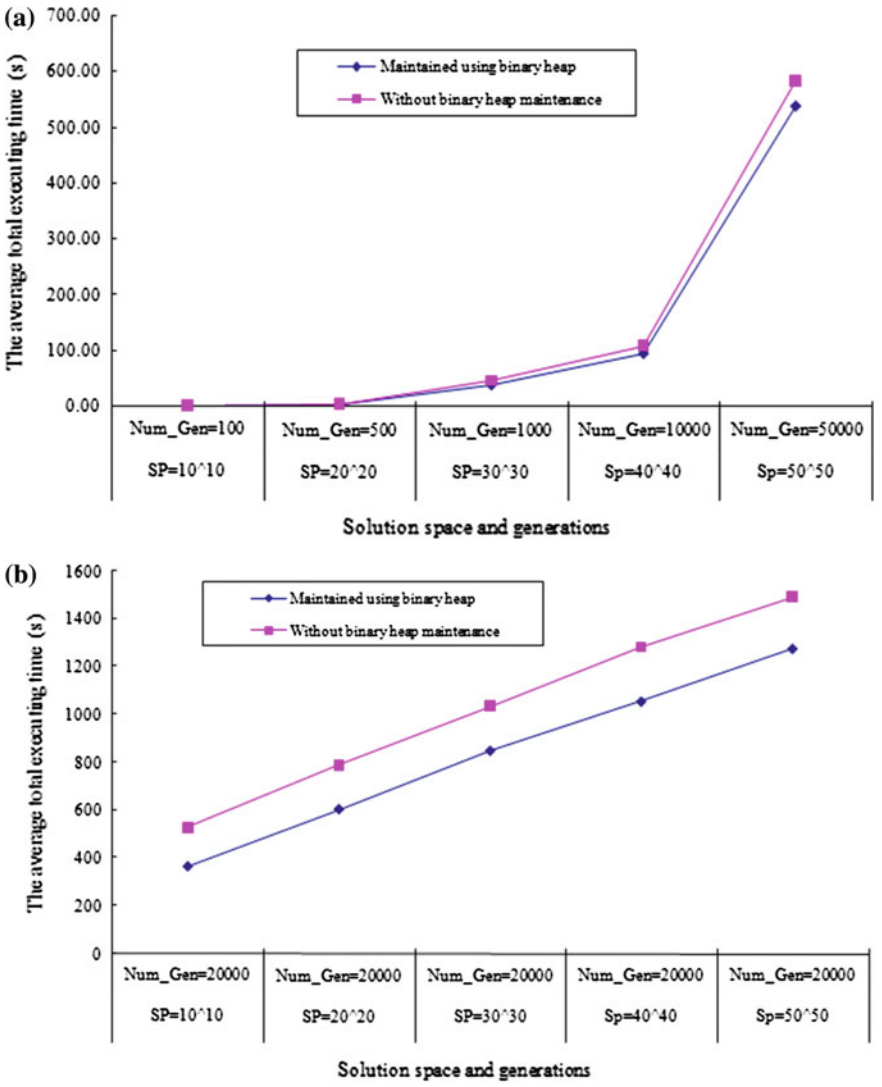


Fig. 6.14 Efficiency of using binary heap to maintain data in GA-BHTR. **a** solutions without binary heap maintenance; **b** solutions with binary heap maintenance.

Table 6.6 Efficiency comparisons between the method using binary heap and Naïve algorithm

Size of turntable	The method using binary heap (s)	Naïve algorithm (s)
100	0.02	0.04
500	0.03	0.18
1,000	0.04	0.36
2,000	0.06	0.74
10,000	0.19	3.66

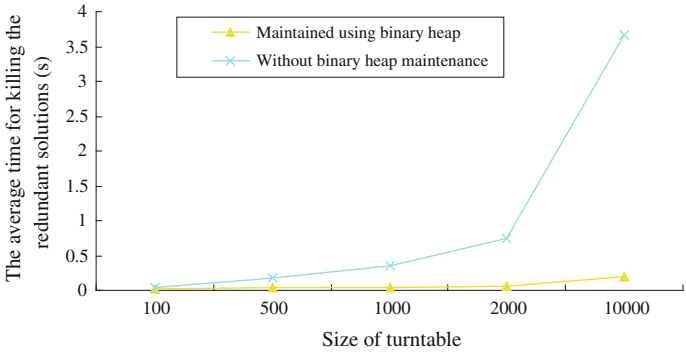


Fig. 6.15 Efficiency comparisons between the method using binary heap and Naïve algorithm

6.5 Summary

In order to address the partner selection problem (PSP) in a virtual enterprise, an improved algorithm named GA-BHTR has been proposed. Several improved strategies, including the transitive reduction, the catastrophe operator and the binary heap operator are implemented and configured as components for algorithm improvements. Combined with the existing operators, i.e., intersection and mutation, the structure with multiple communities is established. Hybridation among traditional and new operators can be easily generated. The simulation and experimental results demonstrated that the structure of multiple communities while maximizing the differences among these communities, and the catastrophe operators have good effectiveness for addressing PSPs, and the efficiency of using the binary heap in GA-BHTR is quite apparent.

References

1. Tao F, Qiao K, Zhang L, Li Z, Nee AYC (2012) GA-BHTR: an improved genetic algorithm for partner selection in virtual manufacturing. Int J Prod Res 50(8):2079–2100
2. Zhang Y, Tao F, Laili YJ, Hou B, Lv L, Zhang L (2012) Green partner supplier selection in virtual enterprise based on pareto genetic algorithms. Int J Adv Manuf Technol 67(9–12):2109–2125
3. Feng B, Fan ZP, Ma J (2010) A method for partner selection of co-development alliances using individual and collaborative utilities. Int J Prod Econ 124:159–170
4. Emden Z, Calantone RJ, Droge C (2006) Collaborating for new product development: selecting the partner with maximum potential to create value. J Prod Innov Manage 23(4):330–341
5. Afonso P, Nunes M, Paisana A, Braga A (2008) The influence of time-to-market and target costing in the new product development success. Int J Prod Econ 115(2):559–568
6. Cowan R, Jonard N, Zimmermann JB (2007) Bilateral collaboration and the emergence of innovation networks. Manage Sci 53(7):1051–1067

7. Bremer CF, Eversheim W (2000) From an opportunity identification to its manufacturing: a references model for virtual manufacturing. CIRP Ann Manuf Technol 49(1):325–329
8. Chen SH, Lee HT, Wu YF (2008) Applying ANP approach to partner selection for strategic alliance. Manag Decis 46(3):449–465
9. Ye F (2010) An extended TOPSIS method with interval-valued intuitionistic fuzzy numbers for virtual enterprise partner selection. Expert Syst Appl 75:7050–7055
10. Drissen-Silva MV, Rabelo RJ (2009) A collaborative decision support framework for managing the evolution of virtual enterprises. Int J Prod Res 47(17):4833–4854
11. Niu SH, Ony SK, Nee AYC (2011) An enhanced ant colony optimiser for multi-attribute partner selection in virtual enterprises. Int J Prod Res 50(8):2286–2303
12. Rocha AP, Oliveira E (1999) An electronic market architecture for the formation of virtual enterprises. In: Proceedings of the IFIP TC5 WG5.3/PRODNET working conference on infrastructures for virtual enterprises: networking industrial enterprises, Porto, Portugal 27–28 Oct, pp 421–432
13. Wu N, Su P (2005) Selection of partners in virtual enterprise paradigm. Robot Comput Integr Manuf 21(2):119–131
14. Chang SL, Wang RC, Wang SY (2006) Applying fuzzy linguistic quantifier to select supply chain partners at different phases of product life cycle. Int J Prod Econ 100(2):348–359
15. Wang TC, Chen YH (2007) Applying consistent fuzzy preference relations to partnership selection. Omega 35(4):384–388
16. Zeng ZB, Li Y, Zhu WX (2006) Partner selection with a due date constraint in virtual enterprises. Appl Math Comput 175(2):1353–1365
17. Ye F, Li YN (2009) Group multi-attribute decision model to partner selection in the formation of virtual enterprise under incomplete information. Expert Syst Appl 36(5):9350–9357
18. Jarimo T, Salo A (2009) Multicriteria partner selection in virtual organizations with transportation costs and other network interdependencies. IEEE Trans Syst Man Cybern Part C Appl Rev 39(1):124–129
19. Ding JF, Liang GS (2005) Using fuzzy MCDM to select partners of strategic alliances for liner shipping. Inf Sci 173(1–3):197–225
20. Huang JJ, Chen CY, Liu HH, Tzeng GH (2010) A multiobjective programming model for partner selection-perspectives of objective synergies and resource allocations. Expert Syst Appl 37:3530–3536
21. Liou JJH, Tzeng GH, Tsai CY, Hsu CC (2011) A hybrid ANP model in fuzzy environments for strategic alliance partner selection in the airline industry. Appl Soft Comput 11(4):3515–3524
22. Famuyiwa O, Monplaisir L, Nepal B (2008) An integrated fuzzy-goal-programming-based framework for selecting suppliers in strategic alliance formation. Int J Prod Econ 113(2):862–875
23. Mukherjee A, Kwon HM (2010) General auction-theoretic strategies for distributed partner selection in cooperative wireless networks. IEEE Trans Commun 58(10):2903–2915
24. Baum JAC, Cowan R, Jonard N (2010) Network-independent partner selection and the evolution of innovation networks. Manage Sci 56(11):2094–2110
25. Hajidimitriou YA, Georgiou AC (2002) A goal programming model for partner selection decisions in international joint ventures. Eur J Oper Res 138(3):649–662
26. Huang XG, Wong YS, Wang JG (2004) A two-stage manufacturing partner selection framework for virtual enterprises. Int J Comput Integr Manuf 17(4):294–304
27. Fischer M, Jahn H, Teich T (2004) Optimizing the selection of partners in production networks. Robot Comput Integr Manuf 20(6):593–601
28. Amid A, Ghodsypour SH, Brien CO (2006) Fuzzy multiobjective linear model for supplier selection in a supply chain. Int J Prod Econ 104(2):394–407
29. Deans I (1999) An approach to the environment management of purchasing in the utilities sector. Eco-Manage 6(1):11–17

30. Crispima JA, de Sousab JP (2010) Partner selection in virtual enterprises. Int J Prod Res 48(3):683–707
31. Saen RF (2007) Suppliers selection in the presence of both cardinal and ordinal data. Eur J Oper Res 183(2):741–747
32. Sari B, Sen T, Kilic SE (2008) AHP model for the selection of partner companies in virtual enterprises. Int J Adv Manuf Technol 38(3–4):367–376
33. Chan FTS, Kumar N, Tiwari MK, Lau HCW, Choy KL (2008) Global supplier selection: a fuzzy-AHP approach. Int J Prod Res 46(14):3825–3857
34. Ip WH, Yung KL, Dingwei W (2004) A branch and bound algorithm for sub-contractor selection in agile manufacturing environment. Int J Prod Econ 87(2):195–205
35. Wang ZJ, Xu XF, Zhan DC (2009) Genetic algorithm for collaboration cost optimization-oriented partner selection in virtual enterprises. Int J Prod Res 47(4):859–881
36. Bu Y, Zhou W, Yu J (2008) A discrete pso algorithm for partner selection of virtual enterprise. In: Proceedings of the 2nd IEEE international symposium on intelligent information technology application. Shanghai, China, pp 814–817
37. Crispim JA, de Sousa JP (2009) Partner selection in virtual enterprises: a multi-criteria decision support approach. Int J Prod Res 47(17):4791–4812
38. Tao F, Zhao D, Hu Y, Zhou Z (2010) Correlation-aware resource service composition and optimal-selection in manufacturing grid. Eur J Oper Res 201(1):129–143
39. Tao F, Zhang L, Zhang ZH, Nee AYC (2010) A quantum multi-agent evolutionary algorithm for selection of partners in a virtual enterprise. CIRP Ann Manuf Technol 59(1):485–488
40. Yeh WC, Chuang MC (2011) Using multi-objective genetic algorithm for partner selection in green supply chain problems. Expert Syst Appl 38:4244–4253
41. Renner G, Ekart A (2003) Genetic algorithms in computer aided design. Comput Aided Des 35(8):709–726
42. Toledo CFM, França PM, Morabito R, Kimms A (2009) Multi-population genetic algorithm to solve the synchronized and integrated two-level lot sizing and scheduling problem. Int J Prod Res 47(11):3097–3119
43. Li Y, Zhang S, Zheng X (2009) Research of multi-population agent genetic algorithm for feature selection. Expert Syst Appl 36(9):11570–11581
44. Kapanoglu M, Koc IO (2006) A multi-population parallel genetic algorithm for highly constrained continuous galvanizing line scheduling. Lect Notes Comput Sci 4030:28–41
45. Simplesource (2007) Genetic algorithm for addressing TSP. http://simplesource.blog.163.com/blog/static/10341406200761041303 12/2007.7. Accessed 25 Mar 2010

Chapter 7
CLPS-GA for Energy-Aware Cloud Service Scheduling

In this chapter, CLPS-GA (A Case Library and Pareto Solution-based improved Genetic Algorithm) [1] for addressing Energy-aware Cloud Service Scheduling (ECSS) in cloud manufacturing is introduced. With the modeling of cloud service scheduling in distributed integrated manufacturing system, a multi-parent crossover operator and a case library for searching is designed. Both of them are based on the general configurable population-based I/O. In terms of the Pareto searching procedure, multi-parent crossover operator is programmed based on the original single-point crossover operator and encapsulated as a new component. For improving searching diversity, a case library can be constructed based on the existing GA class with a new storage array and a new case handling operator. Moreover, based on existing operators of genetic algorithm, a two stage algorithm structure is established.

7.1 Introduction

Cloud computing is a new technique based on distributed computing, parallel computing and grid computing. Numerous implementation plans from various major software enterprises have been proposed, including the Amazon Web Services (AWS, Amazon Web services), Google's App Engine cloud computing platform, IBM's blue cloud plan, and Microsoft's software service (SS) [2]. With the introduction of virtualization, the processors are no longer dedicated to a single task, but can be shared by multiple users while creating the illusion that each separate execution environment is running its own private computer. This has definitely improved the overall system throughput as well as its infrastructure resource utilization. Yet it has also resulted in a more massive, diverse and heterogeneous resource environment. Now the issue of resource scheduling and management has become more complex and difficult than ever.

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_7

For cloud computing services management centers, as shown in Fig. 7.1, the concern is not only providing the best quality of service (QoS), usually measured by task completion time, implementation reliability, etc., but also reducing the total cost, among which the energy consumption has gradually become a significant component. According to a recent report published by the European Union, a 15–30 % decrease in carbon emission is required to keep the increase in global temperature under 2 centigrade by the year 2020. Gartner, in April 2007, estimated that the information and communication technology (ICT) generated about 2 % of the total global dioxide carbon emissions, which is tantamount to the aviation industry [3]. Bianchini and Rajamony [4] also confirmed that the operation of cloud management center require high energy usage. Today, a typical management center with 1,000 racks needs 10 MW of power to operate [3]. This would inevitably induce high electricity charges. Therefore, to reduce the carbon emission in addition to the operation cost, more and more scholars tend to consider energy consumption as one of optimization indexes during the resource management. Therefore, the resource scheduling problem in cloud computing is better to be formulated as a multi-objective optimization (MOO) problem.

Resource scheduling has been proved as a NP-complete problem [5]. Traditional deterministic optimization algorithms demonstrate limited capability in dealing with NP-complete problems because of the combinatorial explosion encountered when the data size is large. In recent years, researchers show much interest on artificial intelligence methods such as evolutionary computation, especially genetic algorithm (GA). Owing to its implicit parallelism and intelligence, GA has been applied to solve some large-scale, nonlinear resource scheduling in clusters or grid systems [6], and has achieved good results. However, in a world with combination of cloud computing and virtualization, traditional GA inevitably meets its own limitations when it is employed for scheduling ultra-large-scale virtual resources: low search speed, risk of falling into a local optimum and far-from-best use of its parallel mechanism.

In addition, the traditional method of solving a MOO problem is to weight the relative degree of importance of each target and then transform it into a single objective optimization (SOO) problem. For resource scheduling models in cloud computing, this method specifically has two major drawbacks. First, users are charged on a pay-per-use basis [7, 8] and they often want to choose from several solutions; while this method can only provide one. Secondly, the scheduling result is highly sensitive to the values of weighted parameters so that the decision-makers must acquire a full and comprehensive knowledge of the problem. However, it is impossible to obtain a generalized set of parameters because different users may have different needs. Finding out all the possible non-inferior (or non-dominated) solutions for the decision-makers to select would make more sense. Taking advantage of the strong global search ability of GA, the solution set distributed in the Pareto front could be identified. In the meantime, extra efforts should be made to maintain the diversity of the population.

In a nutshell, this article studies the scheduling of cloud computing resources with the objectives to minimize both the makespan and energy consumption and

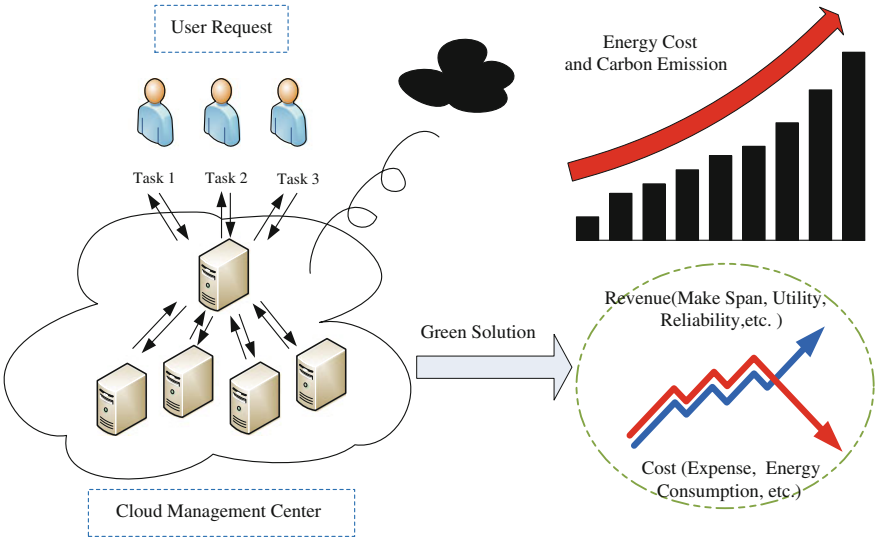


Fig. 7.1 Energy-aware cloud scheduling

solves the resource scheduling problem by proposing a hybrid approach to find the set of Pareto front solutions. The primary contributions of this paper include:

(1) Establishing a model of resource scheduling in a highly heterogeneous cloud environment with uncertain load information in each processor.
(2) Proposing a new hybrid genetic algorithm approach composed of a case library (CL) and a multi-objective genetic algorithm (GA) to find the set of Pareto-front solutions, hence called CLPS-GA. The major components of CLPS-GA include a multi-parent crossover operator (MPCO), a two-stage algorithm structure, and a case library.
(3) Verifying the effectiveness of CLPS-GA, specifically the role of MPCO on solutions' diversity and quality, and that of case library on algorithm's convergence and stability and comparing with other strategies in GA through experimental simulations.

7.2 Related Works

The operation of a cloud computing system can be divided into five stages: *user request*, *resource exploration*, *resource scheduling*, *service and process monitoring* and *returning feedback*, in which the third one is the most important part because it directly influences the final quality of services and the total cost during the process. The users' requests in cloud service are commonly represented as a directed acyclic graph (DAG). Deelman et al. [9] have done considerable work on the planning,

mapping and data-reuse in the area of DAG scheduling. The Pegasus, proposed by him, has become a widely used framework that maps complex scientific workflows onto distributed resources such as the Grid. Other well-known projects in DAG mapping include GridFlow [10], ICENI [11], GridAnt [12], Triana [13] and Kepler [14], most of which are based on earliest finish time, earliest starting time or the high processing capabilities. So basically, the resource is selected according to its performance.

Recently, as discussed before, from both economic and ecological perspectives, energy consumption by Cloud infrastructures has become a key concern for cloud management center. Mayo and Parthasarathy [15] observed that even simple tasks such as listening to music can consume significant different amount of energy on a variety of heterogeneous devices, and suggested the service providers to pay attention to deploy software on right kind of infrastructure which can execute the software most efficiently. One of the first works that dealt with performance and energy trade-off was by Chase et al. [16], in which a bidding system to deliver the required performance level and switching off unused servers was proposed. Kephart et al. [17] addressed the power-performance tradeoffs using a utility function approach in a non-virtualized environment. Beloglazov [18] redefined the architectural principles of power management in virtualized heterogeneous environments and proposed a more holistic approach on machine status switching. Other popular techniques that help reducing power consumption in virtual machines (VM) include VM consolidation [19] and VM migration [20]. However, data deployment of each virtual machine within a Cloud management center can be really hard to maintain. Thus, various indirect load estimation techniques must be used before most energy-aware schemes are implemented.

As far as algorithm is concerned, the mapping of tasks to computing resources is an NP-complete problem in the general form. Traditional deterministic scheduling methods cannot achieve good results in cloud scheduling problems due to the potential combinatorial explosion. Meanwhile, more and more artificial intelligent algorithms have been employed to solve the scheduling problems. Lei and Xiong [21] proposed an effective GA to minimize the expected makespan and the expected total tardiness, and confirmed that it outperformed the traditional dispatching rules. Jin et al. [22] studied two metaheuristic algorithms with the objective to minimize the makespan based on shop partitioning and simulated annealing for multistage hybrid flow shop scheduling problems, and the proposed approaches had been implemented in a real-life printed circuit board assembly line. Tang et al. [23] proposed a neural network model and algorithm for dynamic hybrid flow shop scheduling problem with the objective to minimize average flow time, or average tardy time, or percentage of tardy jobs. Other attempts on scheduling problems include tabu search by Ishibuchi et al. [24], particle swarm optimization by Pandey et al. [25], ant colony optimization by Niu et al. [26], etc.; most of which study the standard single-objective optimization. Li and Li [27] considered three QoS criteria for scheduling on the grid, namely payment, deadline and reliability, and formulated them as utility function, yet still a variation of single-objective optimization.

In the past decade, scholars have been working on finding the Pareto front for MOO problems, of whom the vast majority have been dedicated to multi-objective evolutionary algorithms (MOEA). Knowles and Corne [28] proposed the Pareto Archived Evolution Strategy (PAES) algorithm, and proved it to be a nontrivial algorithm capable of generating diverse solutions in the Pareto optimal set. Coello Coello and Pulido [29] addressed the MOO by Micro-Genetic Algorithm (MOGA), where the population memory and external memory are in corporate to both diversify the search space and archive the non-dominated Pareto solutions. Furthermore, the Multi-objective particle swarm optimization (MOPSO) is another class of MOEAs that has been addressed by Coello Coello et al. [30] and Mostaghim and Teich [31]. There are other MOO algorithms, which include multi-objective simulated annealing (MOSA) by Nam and Park [32], multi-objective ant colony optimization (MOACO) by Garcia-Martinez et al. [33], multi-objective memetic algorithm (MOMA) by Chi-Keong et al. [34], etc. So far, MOGA and MOPSO have been proven to have more efficient searching ability and thus are more likely to obtain Pareto solutions in MOO. Using discrete numbers on encoding to correlate chromosome's gene to task-resource mappings, MOGA is more suitable in cloud scheduling compared to MOPSO.

However, classic GA sometimes encounters problems of low convergence rate, premature convergence or other issues especially when dealing with high-dimensional and large-size data. For this reason, many hybridizations have been proposed, including adaptive genetic algorithm (AGA) [35, 36], chaos genetic algorithm (CGA) [37, 38], and local genetic algorithm (LGA) [39, 40]. All of the above-mentioned hybrid algorithms have more or less improved adaptability, local search ability or global search ability of the algorithm. For the fine-tuning of GA parameters, fuzzy logic controller (FLC) has been suggested by Gen and Cheng [41] to regulate crossover ratio, mutation ratio. And Orhan Engin [42] examined the performances of various reproduction, crossover and mutation operators and rates and explored the best values using full factorial experimental design. Based on their work, a new hybrid approach called CLPS-GA, which include a multi-parent crossover operator, a two-step algorithm structure and concept of case library and similarity, is proposed and tested in this paper.

7.3 Modeling of Energy-Aware Cloud Service Scheduling in Cloud Manufacturing

In this paper, the cloud scheduling environment is considered to be highly heterogeneous and includes various processors of uncertain production load information. The scheduling objectives are multiple. Specifically, the focus is on two objectives, minimizing the makespan of tasks and energy consumption. The aim is to find the Pareto set of such MOO problem under the considered environment.

7.3.1 General Definition

The resource scheduling center is considered to have two pieces of information: a collection of user requests and processor information. Each user request is represented by a DAG, which captures a number of task units involved, each unit's own properties, and the relationships among task units. One important property of each task unit that we must take into account for assignment is the task type. For example, a CPU-bounded task will spend most of its time on computing. Thus it will be better assigned to processors with multiple cores or large RAM size. On the other hand, an I/O bounded task mainly deals with peripheral devices; so it might require processor having a large buffer and sufficient external frequency, or bandwidth. Other properties of task unit might include input and output data size, also called the scale of task unit, indicating how much resources it will need from the processor. Besides, there might be dependencies among task units, meaning that the execution of one task unit might depend on the completion of a certain set of task units. Figure 7.2 gives an example of DAGs, in which each node represents a task unit, the color of the node represents its task type, each directed line between two nodes represents their dependency relationship, and we can add weight to the edges to depict the flow size. To give a mathematical formulation of DAG (or user request), it can be roughly denoted as $\mathbf{G} = (\mathbf{V}, \mathbf{T}, \mathbf{E}, \mathbf{DIN}, \mathbf{DOUT})$. The semantics of each parameter are explained as follows.

User Request:

- $\mathbf{V} = \{V_i | i = 1 : n\}$ represents the decomposed task units of each user request, where n is the total number of task units.
- $\mathbf{T} = \{T_i | i = 1 : n\}$ denotes the task type of each unit in $\mathbf{V}$, where $T_i \in \{1, \ldots, T_{\max}\}$ with $T_{\max}$ indicating the total number of task types.
- $\mathbf{E}(n \times n)$ denotes dependencies between task units in $\mathbf{V}$. Let $E_{ij} = 1$, if data obtained from V_i is used by V_j. *Otherwise*, $E_{ij} = 0$.
- $\mathbf{DIN}(1 \times n)$ represents each task unit's input data size.
- $\mathbf{DOUT}(1 \times n)$ represents each task unit's output data size.

As mentioned before, the virtual resource pool is highly heterogeneous: the processors in it can be a server, a work station, or even a remote PC. Even for processors of the same type, saying two servers, their configurations can be quite different. The immediate result can be a substantial variation in performance even when they are handling the same task. But in general, we can use the processor capacity and channel capacity to characterize such heterogeneity in resources. Processor capacity defines how fast a task can be processed by a certain processor, which is directly related to the CPU power and random access memory (RAM) size. It also defines the corresponding cost for processing; in our study, this cost refers to the energy consumption. Similarly, channel capacity defines the rate and cost of communication between two processors. Apparently, channel capacity does not differentiate the task type, since it only deals with data flow. The resource information, M, can be represented by a set of parameters as $\mathbf{M} = (\mathbf{P}, \mathbf{TP}, \mathbf{S}, \mathbf{EP}, \mathbf{DC}, \mathbf{EC})$ with each parameter explained below.

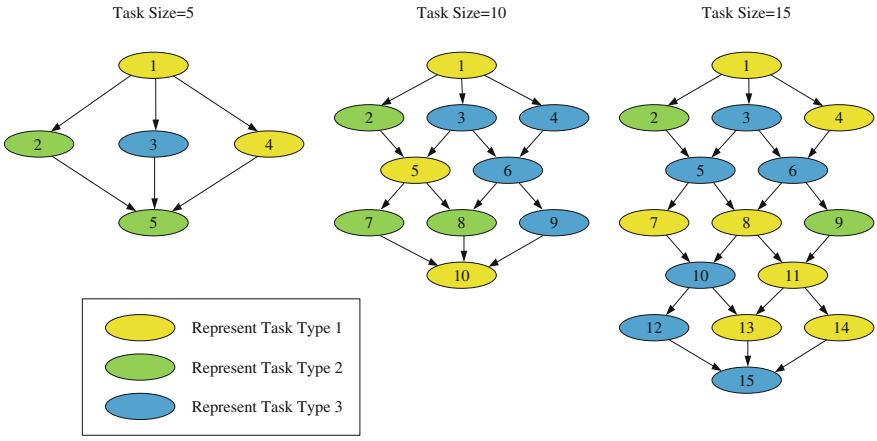


Fig. 7.2 DAGs of user requests

Resource Information:

- $\mathbf{P} = \{P_i | i = 1 : p\}$: represents a collection of processors, where p is the total number of processors.
- $\mathbf{TP}(T_{\max} \times p)$: denotes the computing power of the processor, where TP_{ik} represents time cost for processor P_k to execute the task unit of type i. $\overline{TP}_k$ denotes the average power of processor k, whose value can be obtained by calculating the mean of elements in column k of matrix **TP**.
- **S**: denotes the memory size of each processor.
- $\mathbf{EP}(T_{\max} \times p)$: denotes the computing energy consumption rate, where EP_{ik} represents the energy consumed on processor P_k by executing task unit of type i per unit time per unit data.
- **DC**: denotes the bandwidth between processors, where DC_{kl} represents the transferring rate of data from processor P_k to processor P_l.
- **EC**: denotes the communication energy consumption rate, where EC_{kl} represents the energy consumed by transferring data from processor P_k to processor P_l per unit time per unit data.

Mapping Variables:

- X: denotes the mapping between task units and processors. $X(i) = k$ means that task unit V_i has been assigned to processor P_k to be executed. For $\forall i \in \{1, 2, \ldots, n\}$, $X(i) \in \{1, 2, \ldots, p\}$.

7.3.2 Objective Functions and Optimization Model

As mentioned before, minimizing both the makespan and energy consumption are selected as the two objectives in this study of resource scheduling in cloud manufacturing. The two objectives are contradictory in nature, which mainly come from two aspects:

- Heterogeneity in resources: the fastest resource is not necessarily the cheapest.
- Mechanism of parallelism: makespan is reduced at the cost of more frequent inter-processor communication, which in turn increases the total energy consumption.

The two objective functions are first formulated mathematically below.

(1) Makespan

The makespan is defined as the duration from the moment a user submits his request to the completion of the last task unit. It usually involves waiting time and processing time. We will first calculate the processing time of the user request.

For the decomposed task units of each request, we need to perform a topological sorting to make sure that every task unit can only be dependent on those with smaller indexes. In this way, the total processing time is tantamount to the completion time of task unit V_n. For each task unit V_i, its completion time $TComplete(i)$ can be calculated by adding the latest time for all the needed data to arrive at the current processor and the execution time for the current task unit. Take the first DAG in Fig. 7.1 as an example, if the completion times of task units V_2, V_3 and V_4 are known, we will be able to determine when all the input data for task unit V_5 will arrive. Adding the processing time of V_5, we can obtain the completion time of it. Mathematically, the completion time for task unit V_i can be expressed as

$$TComplete(i) = \max_{j=1:i-1}\{E_{ji} \times TComplete(j) + \frac{E_{ji} \times DOUT_j}{DC_{X(j)X(i)}}\} + TP_{T_iX(i)}DIN_i \tag{7.1}$$

The values of elements in vector **TComplete** can be obtained recursively. If the waiting time is ignored, we can claim the value of $TComplete(n)$ to be the makespan of the user request, where n is the last task unit of the user request of concern.

However, Eq. (7.1) holds only when the processor is dedicated to the task unit by which it is assigned. But with virtualization, the fundamental idea is to abstract the hardware of a single computer into several different execution environments, creating an illusion that each separate execution environment is running its own private computer. Therefore, you think you own the CPU, but the ownership is actually switching back and forth among different users. Similarly, you think you have the whole memory, yet in fact it is just a virtual memory and you still need to

swap in and out to get the necessary codes and data into the actual physical memory. The above two points imply that the degree of multi-threading cannot be too high, otherwise the CPU would spend quite an amount of time on context switch and page fault, and worse still, thrashing might happen. Since the degree of multi-threading cannot be too high, if too much work have been assigned to a certain processor, some of them need to queue up awaiting the CPU, which adds up the waiting time. Therefore, the balance of load distribution among processors is particularly important. However, the major difficulty in achieving the absolute load balance lies on the lack of current load information of each processor. Though such information can be measured, the resource providers will not make it public to management center, and they tend to understate it so as to assume more tasks. This situation forces us to find another way to get around this problem. Though we cannot master the information on user requests that have already been assigned, we can control the load distribution of the user request to be assigned. It is restrictive, but effective. In this paper, it is believed that the ideal ratio of load distribution should depend on the memory size and the average computing power of each processor according to Netto and Buyya [43]. Thus, the load balance is defined as:

$$LoadBalance = \sum_{k=1}^{p} (LoadPortion(k) - BestPortion(k))^2 \tag{7.2}$$

where,

$$LoadPortion(k) = \frac{\sum_{i=1}^{n} DIN_i | X(i) = k}{\sum_{i=1}^{n} DIN_i} \tag{7.3}$$

$$BestPortion(k) = \frac{S_k / \overline{TP}_k}{\sum_{k=1}^{p} S_k / \overline{TP}_k} \tag{7.4}$$

It is assumed that the initial load distribution on processors satisfies the ideal ratio, and any deviation from this ratio caused by the current assignment will run a risk that some processors might become busy, forcing some tasks to be placed into the waiting queue, in turn leading to a prolonged makespan. Therefore, Load-Balance can be considered as a risk parameter that could influence the makespan. Accordingly, the final makespan is defined as:

$$FinalTComplete = TComplete(n) \times e^{\alpha \cdot LoadBalance} \tag{7.5}$$

In Eq. (7.5), α is a parameter used to indicate the importance of load balance. When the access requests are high and data traffic flow is heavy, a large α is set to represent the possible delay on makespan caused by imbalance in load distribution. While the network is idle, α takes a value of zero, which means that the impact of *LoadBalance* on makespan can be ignored.

(2) Energy consumption

The energy consumption is defined as all the power used by every pieces of hardware during the period of fulfilling a user request. The analysis on energy consumption carried out by Beloglazov [18] reveals that CPU consumes the main part of energy compared with memory, disk storage and other I/O interfaces. Specifically for CPU, energy consumption ratio mainly depends on its voltage and frequency, which means, as long as the working state of CPU is fixed, the energy consumption ratio will remain unchanged, as expressed in matrices **EP** and **EC**.

Equation (7.6) gives the mathematical formula of the total energy consumption, which comprises of two parts: computing energy consumption and communication energy consumption. Each part can be further computed as the product of energy consumption ratio, time span and data size, as shown in Eqs. (7.7) and (7.8), respectively.

$$EnergyConsumed = EnergyComp + EnergyComm \tag{7.6}$$

$$EnergyComp = \sum_{i=1}^{n} EP_{T_iX(i)} TP_{T_iX(i)} DIN_i \tag{7.7}$$

$$EnergyComm = \sum_{i=1}^{n} \sum_{j=1}^{i-1} \frac{E_{ji} DOUT_j}{DC_{X(j)X(i)} / EC_{X(j)X(i)}} \tag{7.8}$$

7.3.3 Multi-Objective Optimization Model for the Resource Scheduling Problem

Based on the above descriptions, the objective functions and constraints of the problem can be represented as follows.

(1) Objective functions:

Objective functions can be written as $Min(FinalTComplete)$ and $Min(EnergyConsumed)$, and based on equations given in Eqs. (7.1)–(7.8), the formulas of these two objectives can be rewritten as:

$$Min\left(TComplete(n) \times e^{\alpha \cdot \sum_{k=1}^{p} \left(\frac{\sum_{i=1}^{n} DIN_i | X(i)=k}{\sum_{i=1}^{n} DIN_i} - \frac{S_k / \overline{TP}_k}{\sum_{k=1}^{p} S_k / \overline{TP}_k} \right)^2} \right) \tag{7.9}$$

$$Min\left(\sum_{i=1}^{n} EP_{T_iX(i)} TP_{T_iX(i)} DIN_i + \sum_{i=1}^{n} \sum_{j=1}^{i-1} \frac{E_{ji} DOUT_j}{DC_{X(j)X(i)} / EC_{X(j)X(i)}} \right) \tag{7.10}$$

(2) Constraints:

The mapping variable X defined in Sect. 3.1 can already ensure that the number of selected services is equal to the number of decomposed task units in the user request and only one processor is selected for each task unit. Apart from this, there are other factors to consider.

- Maximum processing time *MaxTProcessing* for each task unit.
- Maximum communication time *MaxTCommunication* for each task unit.

For sake of fairness, there should be an upper bound for how long a single task unit can be allowed to hold a processor or channel. If such a maximum time is reached, it may suggest that the task is inappropriately assigned and should probably be re-assigned. Or it is deemed as a "giant" task.

- Maximum processing energy consumption *MaxEProcessing* for each task unit;
- Maximum communication energy consumption *MaxECommunication* for each task unit;

Similarly, there should be an upper bound for how much energy a single task unit can be allowed to consume. Neither an inappropriate assignment nor a "giant" task is acceptable.

- Acceptable range for load portion [0, *UpperLPortion*(k)] for processor.

To balance the load distribution, the management center sometimes set a range on how large the portion of a user request can be assigned to a certain processor. The lower bound is usually 0, and the upper bound on different processors can be varied, mainly depending on the processor capacity.

Accordingly, the following constraints can be obtained:

$$X(i) \in \{1, 2, \ldots, p\},\ \forall i \in \{1, 2, \ldots, n\} \tag{7.11}$$

$$TP_{T_iX(i)} DIN_i \leq MaxTProcessing \tag{7.12}$$

$$\max_{j=0:i-1} \left\{ \frac{E_{ji} \times DOUT_j}{DC_{X(j)X(i)}} \right\} \leq MaxTCommunication \tag{7.13}$$

$$EP_{T_iX(i)} TP_{T_iX(i)} DIN_i \leq MaxTProcessing \tag{7.14}$$

$$\sum_{j=0}^{i-1} \frac{E_{ji} DOUT_j}{DC_{X(j)X(i)} / EC_{X(j)X(i)}} \leq MaxECommunication \tag{7.15}$$

$$\frac{\sum_{i=1}^{n} DIN_i | X(i) = k}{\sum_{i=1}^{n} DIN_i} \leq UpperLPortion(k) \tag{7.16}$$

Equation (7.11) ensures that each task unit can only select one processor from the virtual resource pool, and Eqs. (7.12)–(7.16) give constraints from the aspects of

MaxTProcessing, *MaxTCommunication*, *MaxEProcessing*, *MaxECommunication* and *UpperLPortion(k)*, respectively.

In summary, the resource scheduling problem in cloud computing has been formulated as a MOO problem subject to various constraints. In the next section, the proposed CLPS-GA algorithm aimed at finding Pareto solutions for MOO problems will be described.

7.4 Cloud Service Scheduling with CLPS-GA

This section starts with a brief review on multi-objective combinatorial optimization and genetic algorithms. Then, a new case library and Pareto solution-based improved Genetic Algorithm (CLPS-GA) is established.

7.4.1 Pareto Solutions for MOO Problems

The objectives in a MOO problem are normally contradicted. When achieving one optimal objective, the other objectives may be affected and get worse. Unlike SOO, which has a unique optimal solution, many new concepts have been introduced in solving MOO problems.

7.4.1.1 Domination and Non-Inferiority

In MOO problems, if individual p has at least one objective better than individual q, and all of p's other objectives are no poorer than those of q's. It is said that individual p dominates individual q, otherwise individual p is non-inferior to individual q.

7.4.1.2 Rank, Front and Pareto Solutions

If p dominates q, a lower rank is assigned to p than q. If p and q are non-inferior to each other, they have the same rank value. Individuals with rank 1 belong to the first front, individuals with rank 2 belong to the second front, and the rest can be deduced by analogy. By sorting with rank, individuals can be identified to different fronts. Normally, individuals in the first front are called the Pareto solutions set while individuals not in the first front are dominated solutions. Figure 7.3 graphically shows examples of Pareto solutions (closed circles) and dominated solutions (open triangles) in a bi-objective optimization problem, in which both objective functions are assumed to be minimized.

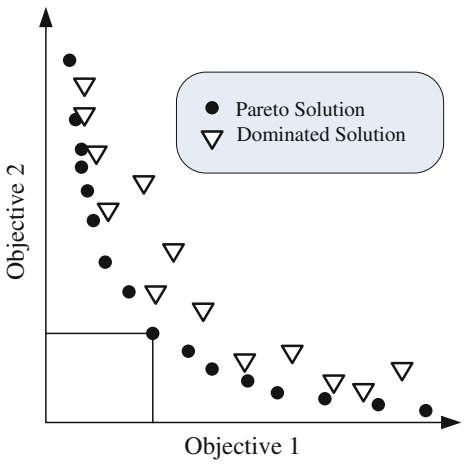


Fig. 7.3 Example of solutions in MOO problem

Crowding Distance:
Crowding distance denoted as *Distance* measures the distance between a particular individual with others in the same front. The formulas are given as follows.

$$Distance(i) = \sum_{j=1}^{NObj} Dis(i,j),\ \forall i = 1 : NInd \tag{7.17}$$

$$Dis(index(i),j) = \begin{cases} \min\left(\inf, data(index_j(i+1),j) - data(index_j(i-1),j)\right), \forall i = 2 : NInd - 1 \\ \inf,\ i = 1\ or\ i = NInd \end{cases} \tag{7.18}$$

$$Score(i,j) = \frac{score(i,j)}{1 + \max_{k=1:NInd,\, score(k,j) \neq \inf}(|score(k,j)|)},\ \forall i = 1 : NInd,\ j = 1 : NObj \tag{7.19}$$

where, *NInd* represents the number of individuals, and *NObj* represents the number of objectives. Equation (7.17) indicates that Distance(i) is the sum of crowding distance of individual i for each objective j denoted as Dis(i, j) whose value can be obtained by Eq. (7.18). While *data* in Eq. (7.18) is the sorted matrix for each column in matrix *Score*, where index$_j$ is the returned index for Column j in *Score* after sorting, and *Score* can be calculated by mapping value of objective j of individual i, denoted as score(i, j), into region (−1, 1), as expressed by Eq. (7.19).

Apparently, the longer the crowding distance, the more difference of objective function values of two neighboring individuals in the front is; thereby the more diverse the population is. Note that only individuals in the same front are needed to calculate the crowding distance; distances between individuals of different fronts are of no significance.
ParetoFraction:

ParetoFraction is defined as a parameter valued between 0 and 1, representing the proportion of the number in the Pareto front out of the whole population. Based on it, the number of individuals in the best front is equal to $\min\{ParetoFration \times PopulationSize, Numbers\ Existed\ in\ the\ Pareto\ Front\}$.

7.4.2 Traditional Genetic Algorithms for MOO Problems

Figure 7.4 shows the framework of GA in solving a MOO problem. Overall, the GA for solving MOO problems appears similar to that for solving SOO problems. Except that some adaptations are required, mainly in the evolution process and in the determination of terminating conditions.

(1) Evolution process

Figure 7.5 shows the structure of evolution operator of GA in addressing MOO problems. By selection, crossover and mutation, a new generation of individuals is generated and evaluated. The fitness values of all individuals can be used to evaluate the rank of each individual and the crowding distance. Then through the trimming operation, the population size maintains stable throughout the evolution process.

a. Selection:

The selection process is often carried out by the tournament selection operator that is based on individual's rank and crowding distance. It not only allows the convergence of the evolution process to the best Pareto front but also maintains some diversity of the potential solutions. To be more specific, individuals with lower ranks have higher chances to be selected regardless of its crowding distance; and between those who have the same rank, the one with larger crowding distance would be more likely to be selected because of its contribution to higher diversity.

b. Trimming population

The number of individuals allowed in the first front can be calculated according to the *ParetoFraction* coefficient and likely, the numbers on other fronts can also be obtained based on certain formulas. By using the tournament selection operator, the trimming process can be effectively done.

c. Termination criteria

The execution of an metaheuristic algorithm is often terminated based on the two conditions given below: (i). The number of iterations exceeds the set maximum; (ii). the cumulative change of function value (*SpreadChange*) of the front individuals is less than a pre-specified tolerance (set as *FunEval* in Table 7.1), and at the same time, Spread(*gen*)is no larger than MeanSpread(*gen*), which means the change of the Pareto front is slow enough so the algorithm has converged.Related values can be obtained as follows.

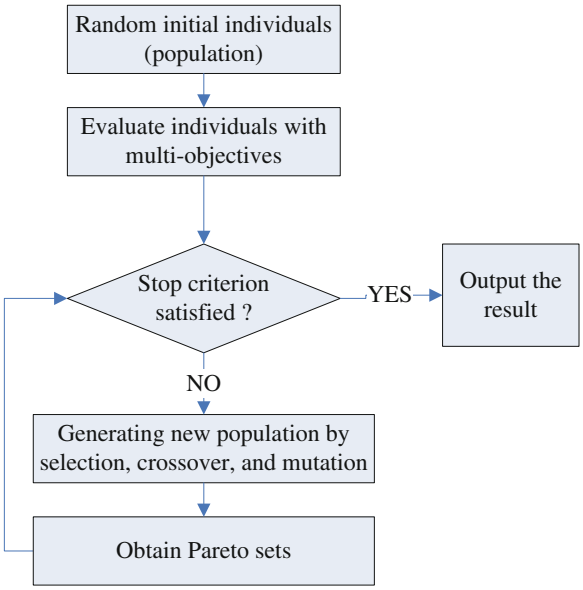


Fig. 7.4 Framework of GA in MOO

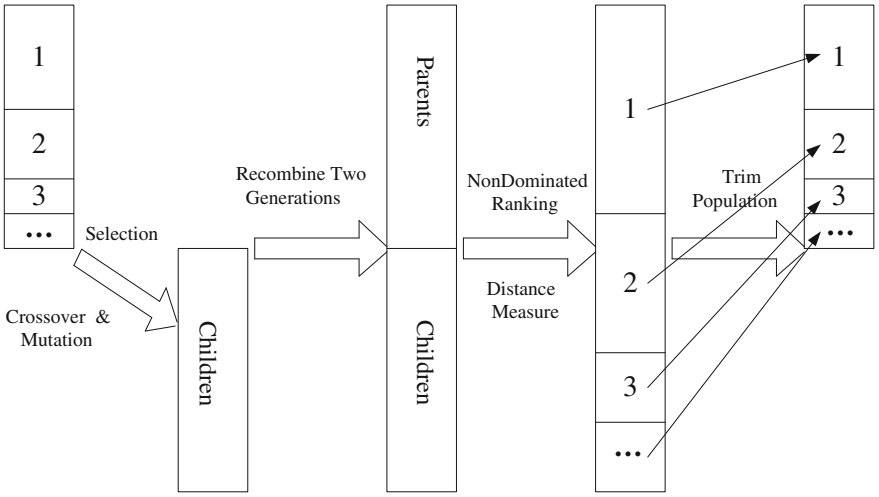


Fig. 7.5 Evolution process of GA in MOO

Table 7.1 Information related to processors and algorithms

Parameters related to the resources	$\mathbf{TP}[3\times 4]=\begin{bmatrix}0.67 & 0.83 & 0.71 & 0.78\\ 0.79 & 0.74 & 0.67 & 0.69\\ 0.72 & 0.80 & 0.69 & 0.67\end{bmatrix}$
	$\mathbf{S}=[1024\ 2048\ 1024\ 2048]$
	$\mathbf{EP}[3\times 4]=\begin{bmatrix}1.08 & 1.01 & 1.05 & 1.03\\ 1.03 & 0.97 & 1.08 & 1.06\\ 1.07 & 1.04 & 1.03 & 1.09\end{bmatrix}$
	$\mathbf{DC}[4\times 4]=\begin{bmatrix}\text{inf} & 5 & 10 & 5\\ 5 & \text{inf} & 5 & 3\\ 10 & 5 & \text{inf} & 10\\ 5 & 3 & 10 & \text{inf}\end{bmatrix}$
	$\mathbf{EC}[4\times 4]=\begin{bmatrix}0 & 0.12 & 0.10 & 0.13\\ 0.12 & 0 & 0.11 & 0.15\\ 0.10 & 0.11 & 0 & 0.10\\ 0.13 & 0.15 & 0.10 & 0\end{bmatrix}$
Parameters related to the algorithm	Population size: 100
	Crossing rate: 0.6
	Mutation rate: 0.3
	ParetoFraction: 0.25
	Maximum generation: 200
	FunEval: $1e^{-3}$

$$SpreadChange(gen)=\begin{cases}\sum_{i=1}^{StallGen}\left(Weight^{StallGen+1-i}\times\frac{|Spread(i+1)-Spread(i)|}{1+Spread(i)}\right), & gen>StallGen\\ \text{inf}, & otherwise\end{cases} \tag{7.20}$$

$$Spread=\frac{extremeParetoDistance+avgDistance}{extremeParetoDistance+NObj\times avgdistance} \tag{7.21}$$

$$extremeParetoDistance=\sum_{i=1}^{NObj}\|extremeParetoSOld(i)-extremeParetoSNew(i)\| \tag{7.22}$$

$$avgdistance=\frac{\sum_{i=1}^{NInd}Distance(i)}{NInd} \tag{7.23}$$

$$avgdistance=\frac{\|Distance-avgDistance\|}{\sqrt{NInd}} \tag{7.24}$$

$$MeanSpread(gen)=\frac{\sum_{i=gen-StallGen}^{gen}Spread(i)}{StallGen+1},\ gen>StallGen \tag{7.25}$$

In Eq. (7.20), *StallGen* is a positive integer, and the algorithm stops if there is no improvement in the objective functions for *StallGen* consecutive generations. *Weight* is a parameter to indicate the impact of *StallGen*, and its value is usually set to be 0.5. The value of *Spread* in each generation can be obtained from Eqs. (7.21) to (7.24), where *gen* represents the number of generations, *NObj* represents the number of objectives, *NInd* represents the number of individuals, *Distance* represents the crowding distance of each individual as defined in Eq. (7.17), *extremeParetoSOld*(i) represents the individual who has the smallest value of objective i in matrix *score* in the last generation, and *extremeParetoSNew*(i) represents the corresponding individual in the current generation. And the value of *MeanSpread* can be calculated according to Eq. (7.25), to be compared with value of *Spread*.

7.4.3 CLPS-GA for Addressing MOO Problems

The classical GA cannot achieve good results in MOO problemsby merely relying on selecting and trimming based on individuals' rank and crowding distance. To better improve the diversity of solutions, the convergence rate, and stability of the algorithm, the CLPS-GA algorithm is proposed with some new improved components. The CLPS-GA is composed of a multi-parent crossover operator (MPCO), a two-stage algorithm, and the concept of case library and case similarity.

(1) Multi-parent crossover operator

Traditional GA usually uses two-parent crossover operator (TPCO). The new MPCO is designed to search in a wider range, thereby increasing the diversity of the population.

The real-coded MPCO typically operates as follows: randomly choose M individuals from the current generation and a new individual X^* is formed by $X^* = \sum_{i=1}^{M} a_i X_i$, where a_i meet the constraint: $\sum_{i=1}^{M} a_i = 1$ and $-0.5 \le a_i \le 0.5$. Our new operator, designed by following the basic idea of real-coded MPCO, is to randomize a group of weighted coefficients and then let the value in corresponding position of individual's chromosome be as close as possible to that of the parent who has the largest coefficient. First, M parents are selected from the population. Then let the new individual be: $X^*(j) = \arg\max\{a(i,j)|i \in \{i, 2, 3, \ldots, M\}\}$, where a is matrix with size $M \times L$, M is the number of parents participating in the crossover operation, L represents the length of chromosome. Each $a(i,j)$ takes on a value between 0 and 1 and can be regarded as the odds for the jth gene in parent X_i to be inherited to the next generation. Accordingly, the corresponding gene in each new individual's chromosome is determined to be the same as that of the parent who has the largest *oddsa*(i,j).

The mechanism of the MPCO operator is illustrated with an example given below. Let the number of parents $M = 4$, and the length of chromosome $L = 8$. Randomize the matrix

$$a = \begin{bmatrix} 0.1783 & 0.3784 & 0.6372 & 0.9883 & 0.4839 & 0.7782 & 0.6782 & 0.0883 \\ 0.7382 & 0.7289 & 0.6228 & 0.8372 & 0.8672 & 0.5772 & 0.3784 & 0.6839 \\ 0.7432 & 0.3648 & 0.1283 & 0.2838 & 0.4836 & 0.1739 & 0.8628 & 0.3784 \\ 0.2738 & 0.9228 & 0.4837 & 0.5738 & 0.7289 & 0.3893 & 0.0384 & 0.5783 \end{bmatrix}$$

Then locate the element with the maximum value in each column of a and return its index, which is [3 4 1 1 2 1 3 2]. Following the index, the gene from the corresponding parent can be found. Figure 7.6 illustrates the multi-parent crossover process, where a new child is generated from Parent One, Parent Two, Parent Three and Parent Four according to the matrix a.

(2) Two-step algorithm structure based on case library

In order to accelerate the convergence speed of GA, a two-stage algorithm structure that makes use of a case library is proposed. The framework of our proposed two-step algorithm is depicted in Fig. 7.7. The two stages refer to, (i) searching for similar cases in the library to help with initialization, (ii) go through evolution as stated previously. Here we first need to make it clear about the definition of similar case.

Similar case: A case in the case library is declared similar to the user request only if the following two conditions are satisfied:

(1) The case must have the same number of task units as the user request.
(2) The similarity function S between the two should be no less than δ, i.e. $S \geq \delta$, where δ represents the threshold. The value of S is related to the type of task unit, the dependency matrix of tasks, and the input and output size of tasks. In our study the impact of data size is ignored and S is defined as:

$$S = \frac{k_t}{n} \times \frac{k_e}{m} \tag{7.26}$$

where k_t represents the number of identical values in the task type vectors (**T**) of the case and the user request, k_e represents the number of same positions of entries with value of 1 in the two dependency matrixes (**E**), n represents the number of task units, and m represents the number of elements with value of 1 in **E** of the user request, namely the number of dependency relationships. For example, the similarity between a user request and one case in the library, with **T**, **E** and **Tcase**, **Ecase** given below, is computed as follows:

$$\mathbf{T} = [1\,2\,2],\ \mathbf{Tcase} = [1\,3\,2],\ \mathbf{E} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},\ \mathbf{ECase} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

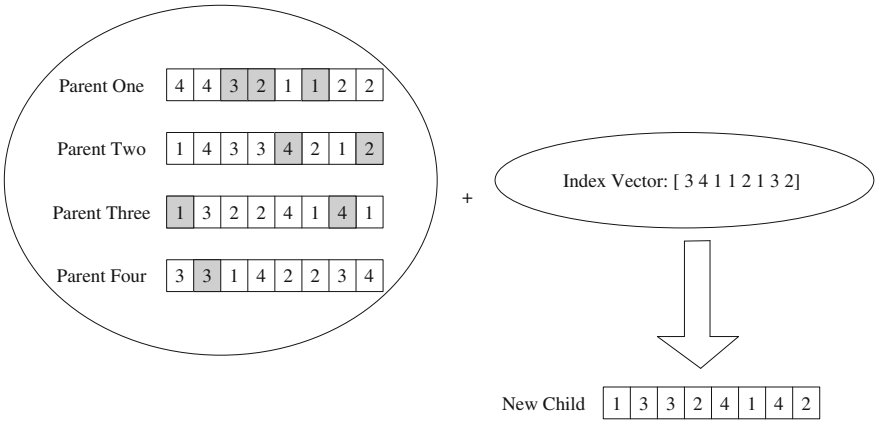
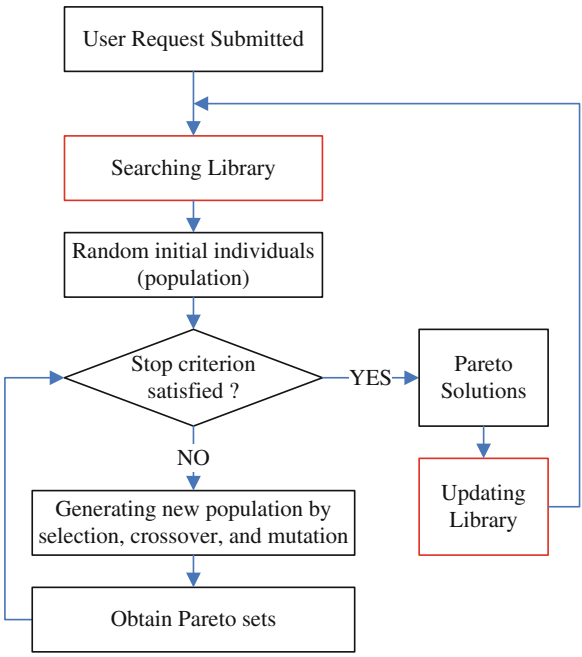


Fig. 7.6 Instance of MPCO

Fig. 7.7 Two-step algorithm structure



According to the above description, it can be determined that $k_t = 2$ and $k_e = 1$, thus the similarity function between them is computed as $S = \frac{2}{3} \times \frac{1}{2} = 0.333$.

Therefore, to calculate the similarity function S between the submitted user request and the cases of the same size, we need to store the task type vector and the dependency matrix for each case in the library. Their corresponding Pareto solutions must also

reside in the library to facilitate with population initialization, if desired. Accordingly, each case in the library can be represented as $\mathbf{case} = \{\mathbf{TCase}, \mathbf{ECase}, \mathbf{Solutions}\}$.

But how can we find the similar cases efficiently? A possible data structure for the case library is suggested below. Knowing that a case with a different size to the user request cannot be a similar case, the cases in the library can be organized into several rows (sequences) according to their sizes. Each sequence is a linked list corresponding to a particular size. The head node stores the number of cases in the sequence, which by the way is not necessarily the same for all sequences. It next points to the first case in the sequence, where the information about $\mathbf{case\_1} = \{\mathbf{TCase}, \mathbf{ECase}, \mathbf{Solutions}\}$ is stored. Then **case_1** will point to the second case labeled as $\mathbf{case\_2} = \{\mathbf{TCase}, \mathbf{ECase}, \mathbf{Solutions}\}$, and continue on until the last case in the sequence is linked. Additionally, an index table will be set up to indicate the address of the head node for each sequence.

To prevent spending too much time on exploring similar cases, an upper bound *Max_Num* for the number of cases in each sequence is set. Consequently, we need to come up with an updating strategy as to which cases should be kept or replaced, given that there are already *Max_Num* cases for one particular sequence while new user requests keep coming in. This issue is dealt with by introducing the concept of concentration function C for each case, which is computed as:

$$C(i) = \frac{1}{N} \sum_{j \in N} Similar(i,j) \tag{7.27}$$

$$Similar(i,j) = \begin{cases} 1, \ S(i,j) > \gamma \\ 0, \ Otherwise \end{cases} \tag{7.28}$$

where, N is the total number of cases in one sequence in the library. S represents the similarity function value between **case_i** and **case_j** according to Eq. (7.26). γ represents the threshold.

If adding new cases will cause violation to the *Max_Num* limitation, then only *Max_Num* cases in these sequence (including the new cases) with lower concentration values are kept. Otherwise, just attach the new cases to the end of the sequence without the need to calculate the concentration.

To better understand the 2-step mechanism, consider the following example. Suppose at most 6 cases is allowed in each sequence, i.e., $Max\_Num = 6$. When a user request comes with task size of 5, it will first visit the index table to obtain the address of the head node corresponding to the 5th sequence. Assume that there are already 5 cases stored in the 5th sequence. The similarity between the user request and each of the 5 existing cases is computed, assuming to be 0.42, 0.77, 0.79, 0.55 and 0.39, respectively. If the threshold δ is set to be 0.8, then no similar cases exist, and hence the initial population shall be randomly generated. But if δ equals to 0.75, both the second and third cases in these sequence are similar to the user request. The Pareto solutions of the third case are retrieved because of its higher similarity. Because coefficient *ParetoFraction* represents the proportion of solutions in the

Pareto front out of the whole population, if it is 0.25, then conducting MPCO three times based on the retrieved solutions in generating the entire initial population. After servicing the request, it can be added to the 5th sequence as a new case, making a total of 6 cases. Now suppose another user request is submitted, and its task size is also 5. This new request will be serviced similarly. Because there are already 6 cases in the 5th sequence, the library will be updated as follows. To maintain the column size of the library, either the new case should be discarded, or an existing case must be selected for replacement. By calculating the concentration values of the 7 cases, supposedly say, 0.50, 0.67, 0.50, 0.33, 0.67, 0.83, and 0.50, respectively, the 6th case is identified to have the highest concentration value and thus should be replaced by the new case. In this way, we can supply the users with more diverse cases while keeping the library size under control.

In detail, the pseudo-code of our proposed algorithm is summarized as follows.

Step 1 Search the case library based on the task size;
Step 2 Does any similar case exist? Yes (go to Step 3)/No (go to Step 4);
Step 3 Retrieve the similar cases with the highest value of similarity to generate the initial population, and go to Step 5;
Step 4 Randomly create the initial population;
Step 5 Evaluate each individual according to the objective functions;
Step 6 Calculate the rank and crowding distance for each individual;
Step 7 Apply the tournament selection;
Step 8 Apply the multi-parent crossover operator (MPCO);
Step 9 Combine the original population and the offspring to create a new population;
Step 10 Apply the trimming operator to maintain the population size;
Step 11 Is the stopping criterion met? Yes (go to Step 12)/No (go to Step 5);
Step 12 Output the Pareto Solutions and update the case library

7.5 Experimental Evaluation

This section describes a series of simulations carried out with the aim to test the performance of CLPS-GA proposed in the last section in solving MOO problems and to verify its effectiveness in comparison other existing algorithms.

7.5.1 Data and Implementation

Experiments were conducted using the Matlab R2009a software platform. First, let's just assume the cloud management center oversees 4 processors. The machine condition can be roughly summarized as follows: processor 1 and processor 2 are old machines, and the remaining two are relatively new. For some reason,

Processor 1 has been set over clocked, so in general, it can process tasks faster than Processor 2, but it consumes more energy. Processor 2 is in good maintenance for years, and its RAM size has recently been extended to 2G. Processor 3 and Processor 4 have employed more advanced hardware chips and more efficient operating systems compared to old machines. Between the two, processor 3 seems to outperform Processor 4, except that it has a smaller RAM size. The energy consumption rate of Processors 3 and 4 seem to be higher than that of Processor 2, but since they are faster, no one knows which processor costs most energy eventually? The above analysis on performance is meaningless if the task type the processor is undertaking is not clearly specified. This example assumes the number of task types to be 3. Strictly speaking, each task type should be defined by explicit numeric values accounting for the percentages of time spent on computing and data transmission. But in this study we simply call them CPU-bounded, I/O bounded, and inter-mediate. Then we need to provide numerical values on processor capacity based on the processor condition and the task type, in the form of matrices **TP**, **S**, and **EP** in Table 7.1. Moreover, it is assumed that the four processors are under full connections, either wireline or wireless, within the same communication subnet. The numeric measurements on channel capacity, i.e., the matrices **DC** and **EC** in Table 7.1, depends on the channel condition, such as the mediums, the power of the base station, interference strength, etc. We really do not plan to go into these details in this study.

Choosing appropriate parameter values is known to have effect in the performance of a metaheuristic algorithm. The best values of basic GA parameters such as *Population Size*, *Crossing Rate*, *Mutation Rate*, and *Maximum Generation* found by Orhan Engin using full factorial experimental design [43] are used. The values of two additional algorithmic parameters related to CLPS-GA, i.e., *ParetoFraction* and *FunEval*, are determined to be those that achieve the best result for our proposed problems in a preliminary experimental study. All algorithmic parameter values used are given in Table 7.1 as well. After that, the proposed CLPS-GA is applied to schedule resources for user requests with increasing number of task units, which are 5, 10, 15, 20, and 30, respectively. Some of their DAGs are depicted in Fig. 7.2. (The task type and dependencies are marked in the DAG; and as far as the input and output data sizes are concerned, it is assumed that each edge in the DAG carries a flow of 10 units).

Our testing experiments and results are organized as follows. First, the performances of TPCO and MPCO are compared and the impact of task size on the convergence, stability and solution diversity of the algorithm is also discussed. Secondly, using the user request with task size of 15 units as an example, cases with different similarity function values are introduced into the initialization process to discuss their performances, to verify the correctness of the expression of similarity function and to determine the similarity threshold. Thirdly, our proposed CLPS-GA is compared with a number of existing enhanced GAs with an example of user request of task size of 15 units and a case with 75 % similarity, and the superiority of CLPS-GA is proved. All results obtained are based on experiments repeated 50 times or more.

7.5.2 Experiments and Results

As mentioned before, the aim of optimization is to provide a set of Pareto solutions for decision-makers to select from. For the bi-objective optimization problem, a two-dimensional plot can be prepared with its two axes representing the two objectives. For example, we can use the vertical axis to denote the makespan, while the horizontal one being energy consumption. The output solutions will be a set of points distributed on the Pareto front (marked in red). Each point actually corresponds to an assignment, and users can visualize which assignment is more suitable according to its position in the plot, or they can just click on a point to obtain the numeric values corresponding to the two objectives, and then make the decision. Matters such as how their request is decomposed, or which task unit has been assigned to which processor are of concern to the cloud management center, but of no much interest to users.

Consider a user request whose DAG is shown as the first one in Fig. 7.2. Even though the task size is only 5, the mapping between the tasks and processors can be very complicated. Assuming that the job is pretty urgent and the user is willing to pay at any cost. One possible solution is simply assigning each task unit to the "fastest" processor according to the task type **T** and matrix **TP**, which means $\mathbf{X} = [1\ 3\ 4\ 1\ 3]$ in this case. But is it the best answer? Employing the classic MOGA produced the results as shown in Fig. 7.8, in which the red curve indicates the final output of Pareto front, while the blue points are some dominated solutions which have been degenerated during the search. The set of Pareto front solutions are given in Table 7.2 with repeated solutions been removed. From the table, one can clearly see that the assignment with the smallest makespan is $\mathbf{X} = [2\ 2\ 3\ 1\ 4]$, and its objective value is (42.111, 54.402). Note that Solution 2has the second lowest value of makespan (42.215 versus 42.111), but consumes less energy. If desirable, one may also consider Solution 3, Solution 4, and so on. In the case that one does not want to consume too much energy because of a very tight budget, Solution 10 is probably the best. However, comparing it with Solution 9, you will find 0.119 more in energy consumption could cut down the makespan by nearly a half. So what would one choose? It really depends on the decision maker!

The above results indicate that the classical MOGA can produce acceptable solutions for the problem. But is there any chance that MOGA can do better? In Fig. 7.8, the blue points do not cross the red curve, which indicate that good solutions get preserved and the bad ones get discarded. But it also implies a possibility of premature convergence. In addition, the solution distribution on the Pareto front needs to be improved. For example, we might wish to obtain more points between Solution 4 and Solution 5. Furthermore, it takes more than 30 iterations for MOGA to finally converge. Several improvements made in this study are presented in the following sections.

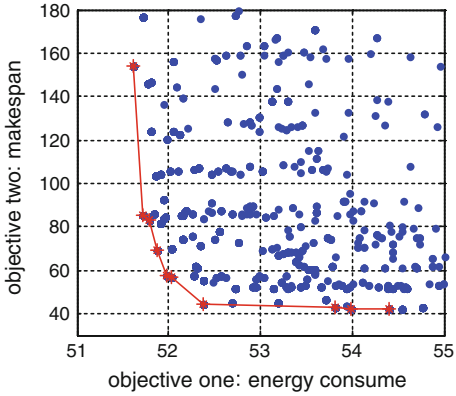


Fig. 7.8 Instance of output pareto plot

Table 7.2 Pareto front of the example

Solution	Mapping variable	Objective value: (Makespan, energy-consumption)
1	[2 2 3 1 4]	(42.111, 54.402)
2	[2 2 1 3 4]	(42.215, 53.982)
3	[1 2 2 3 4]	(42.776, 53.811)
4	[3 4 4 1 2]	(44.262, 52.382)
5	[3 3 4 1 2]	(56.380, 52.032)
6	[1 3 4 1 2]	(57.104, 51.973)
7	[3 2 4 1 2]	(68.883, 51.866)
8	[1 3 4 1 4]	(82.851, 51.781)
9	[1 2 4 1 2]	(85.562, 51.727)
10	[1 2 1 1 2]	(153.831, 51.608)

7.5.3 Comparison Between TPCO and MPCO

Table 7.3 records the mean and variance of average crowding distances and iterations under the two crossover operators with different task sizes based on 50 repeated experimental runs. As can be seen in the table, when the task size is small, 5 or 10 for instance, there is not much difference between the two operators in the mean of average crowding distances. Though MPCO makes it more likely for individuals to inherit genes from a larger range of parents, it does not necessarily mean significant improvements on population diversity. This result is due to the smaller solution space of smaller task size. Under this circumstance, difference between individuals is little, and even less between individuals in the Pareto front. When the parameter *ParetoFraction* is large, the probability in selecting individuals from the Pareto front is higher after trimming the population, which can cause a severe repetition of individuals in the next generation. When the population diversity is low, even carrying genes from multiple parents, the improvement on diversity may not be obvious.

Table 7.3 Experimental data related to TPCO and MPCO

		Mean of average distance	Variance of average distance	Mean of iterations	Variance of iterations
Task number = 5	TPCO	0.0011	8.16×10^{-8}	32.1	876.8
	MPCO	0.0012	4.8×10^{-8}	30.7	652.4
Task number = 10	TPCO	0.0014	2.92×10^{-7}	33.9	467.6
	MPCO	0.0014	2.48×10^{-7}	35.3	448.9
Task number = 15	TPCO	0.0023	1.13×10^{-7}	41.4	528.4
	MPCO	0.0026	2.32×10^{-7}	42.8	456.7
Task number = 20	TPCO	0.0026	7.27×10^{-7}	57.7	572.7
	MPCO	0.0033	4.14×10^{-6}	56.5	648.9
Task Number = 30	TPCO	0.0029	4.38×10^{-7}	83.4	1217.6
	MPCO	0.0036	1.43×10^{-6}	80.8	1012.9

However, when the task size is large, 30 for example, the mean and variance of average crowding distances under MPCO are larger than those of TPCO, specifically 0.0036 and 1.43×10^{-6} for MPCO versus 0.0029 and 4.38×10^{-7} for TPCO. In this case, the influence of our newly designed operator is obvious. A larger average crowding distance means more even distribution of individuals on the Pareto front in the last generation. On the other hand, a small variance means that the improvement on population diversity is stable. The above results, thus, indicate that when the task size is large, MPCO can lead to more diverse individuals on the Pareto front. However, in terms of iteration numbers, it increases with task size and the proposed MPCO does not seem to accelerate the convergence process in all cases when compared to TPCO.

Figure 7.9 shows a typical instance of one experimental result. Five graphs on the left side represent the Pareto front in the last generation under two operators with different task sizes (small to large from top to bottom). Those on the right side represent *SpreadChange* of individuals in the Pareto front throughout the evolution process. The purpose for plotting *SpreadChange* is to check the convergence status of the algorithm. Only when *SpreadChange* is less than the value of *FunEval* in Table 7.1 and *Spread*(*gen*) is no larger than *MeanSpread*(*gen*) according to Eqs. (7.21) and (7.25), one can be certain that the optimal Pareto front has been found. When the task size is small, the two Pareto fronts are very close. For example, when the task size is 5, there is only one different individual in the two curves. But when the task size is large, significant difference between the two curves can be found. One can also clearly see from the graph that curve of MPCO is more close to the axis when task size is large, which reflects a better quality of solutions as the population diversity improves.

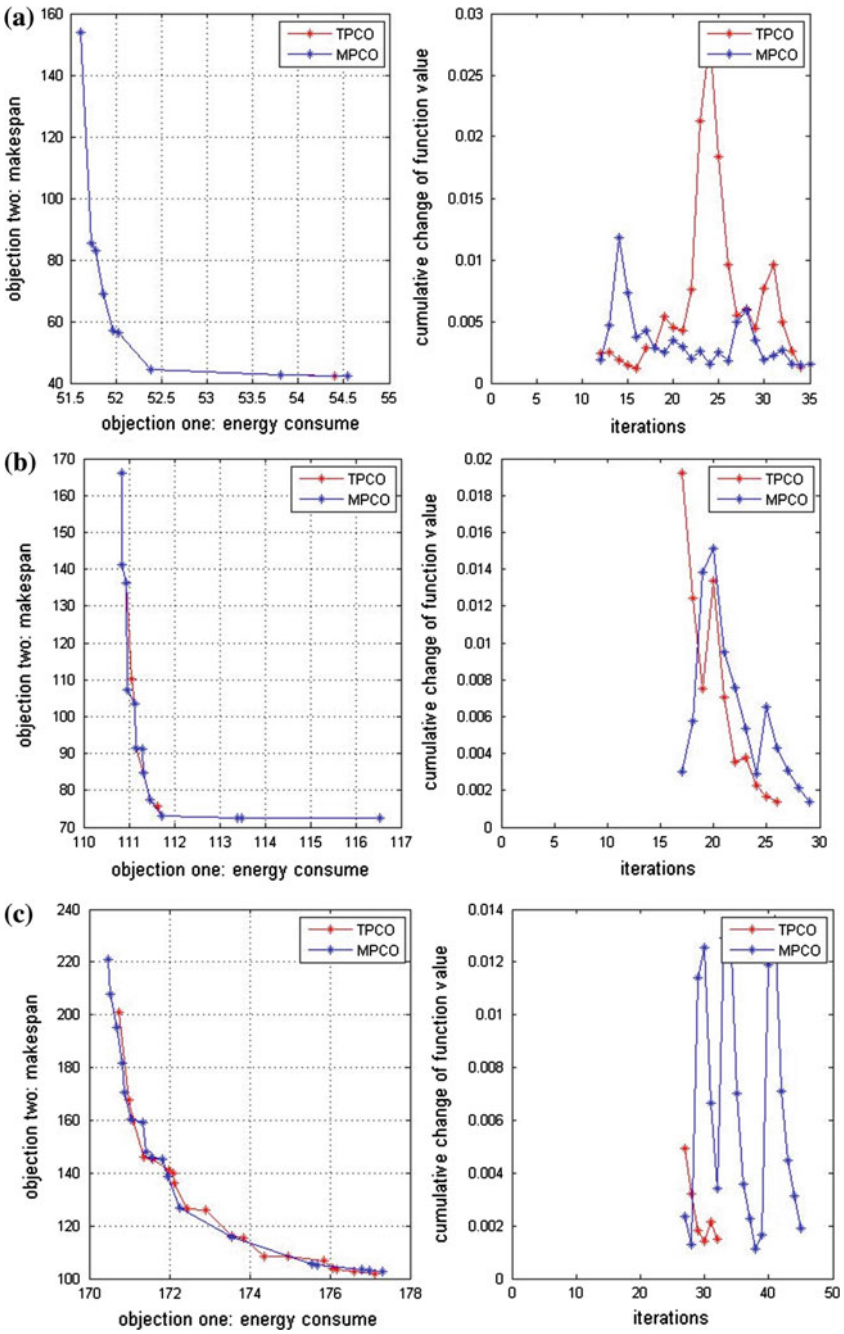


Fig. 7.9 Comparison of TPCO and MPCO as task size increases from 5 (*top*) to 30 (*bottom*)

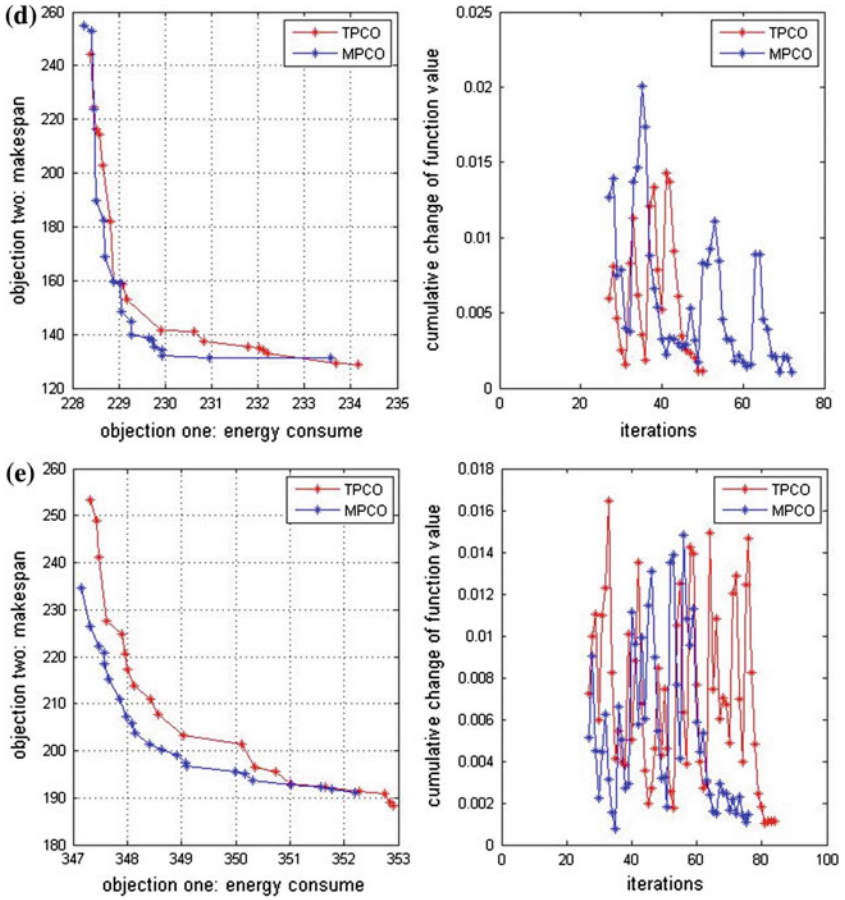


Fig. 7.9 continued

7.5.4 Improvements Due to the Case Library

From the previous analysis, it has been learned that MPCO cannot obviously improve the convergence speed and stability of the algorithm. Being aware that the instability of GA mainly comes from the randomness of initial population, a two-stage framework which makes use of a case library is proposed. By initializing the population with a similar case with close-to-optimal solutions if it exists, it is expected to speed up the convergence process and improve the stability of the algorithm. The similarity threshold is selected based on the performance when cases with different similarities are introduced to the initialization stage of the algorithm. In the experiments, cases with similarity of 100, 75, 50, 25 % (approximately) are chosen to be compared with the baseline of not using the case

library (or equivalently using no similar case). In addition, to better evaluate the effect of case library, the value of *FunEval* is set as 0.0008.

The test results are shown in Table 7.4. The *SpreadChange* in the last generation is used to determine the convergence status of the algorithm. As indicated in the table, when the similarity is equal or greater than 75 %, the iteration number begins to reduce and the reducing amount increases as the similarity increases. The relatively small variance of iteration number also shows that this acceleration on convergence process is stable. This in turn verifies that the expression of the similarity function is correct and useful. However, when the similarity is below 75 %, average iteration numbers do not decrease but instead increase compared to that of the baseline. In other words, introducing cases with low similarity value is not helpful but harmful; it worsens the ability of the algorithm to converge. This result is because the optimal solution set of the current user request is greatly different from that of the cases with low similarity. If initializing with its solutions arbitrarily, it may need to follow a longer route to reach its own Pareto front than random initialization. Therefore, the similarity threshold can be roughly set to be 75 %. It can also be seen from the variances of iteration numbers and *SpreadChange* that, using cases with high similarity to initialize the population improves the stability of the algorithm compared to random initialization.

Figure 7.10 shows the differences in the Pareto front and *SpreadChange* between using cases with 75 % similarity for initialization and using no similar cases at all (the baseline). It can be seen from the Pareto front curves that adopting solutions of similar cases into the initializing stage does not affect the final quality of the individuals. In both scenarios, the individuals are evenly distributed in the Pareto front. The *SpreadChange* curves on the right side clearly indicate the lower variation during the evolution process when case with 75 % similarity is introduced, in sharp contrast to the substantial up-and-downs in early generations with no-case introduced. This difference demonstrates that the evolution process is more stable with the use of similar cases in initialization.

In summary, the practice of introducing cases of high similarity to help with initialization can effectively speed up the convergence rate and improve the evolution stability of the algorithm. The appropriate similarity threshold can be set as 75 %. It should also be noted that the evolution and convergence process could somehow be delayed to some degree if cases with similarity value below the threshold are introduced.

7.5.5 Comparison Between CLPS-GA and Other Enhanced GAs

In this section, the attention is turned to compare the proposed CLPS-GA with some other enhanced GAs such as AGA [35, 36], CGA [37, 38], and LGA [39, 40] in terms of convergence rate, stability and solutions' diversity.

Table 7.4 Experimental data related to case library

	No Similar Case	25 % Similarity	50 % Similarity	75 % Similarity	100 % Similarity
Mean of iterations	48.9	98.3	70.7	36.4	30.5
Variance of iterations	573.5	722.3	582.0	355.5	342.7
Mean of SpreadChange in last generation	7.29×10^{-4}	6.97×10^{-4}	7.42×10^{-4}	7.38×10^{-4}	7.05×10^{-4}
Mean of SpreadChange variance	4.82×10^{-5}	5.13×10^{-5}	3.09×10^{-5}	2.61×10^{-5}	1.88×10^{-5}

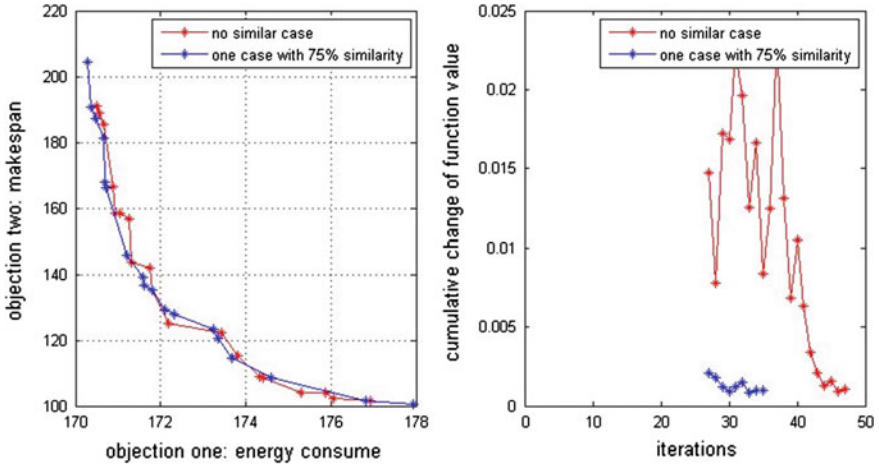


Fig. 7.10 Pareto front curves and *SpreadChange* with and without case library

According to the "no free lunch" scientific theory proposed by Wolpert and Macready [44], no algorithm is able to dominate another in all problems in all aspects. Therefore, any performance improvement on any algorithm might be paid at the cost of time inefficiency or compensated from other aspects. If one considers the convergence rate, stability, solution's quality and diversity as four optimization objectives, then each one of the four algorithms included in the experiment are non-inferior to or non-dominated by others. Table 7.5 records the results of iterations, average distance, and *SpeadChange* to indicate the algorithms' performance on convergence, diversity and stability after at least 50-time experimental repetitions by using AGA, CGA, LGA, and our proposed CLPS-GA, respectively. Figure 7.11 shows one typical instance of Pareto front and *SpreadChange* values as a function of iterations.

After deeper analysis, it can be found that our proposed CLPS-GA does best in algorithm's convergence and stability. By introducing cases with similarity of

Table 7.5 experimental data related to CLPS-GA and other enhanced GAs

Algorithm	Mean of iterations	Variance of iterations	Mean of average distance	Variance of average distance	Mean of *SpreadChange* in last generation	Mean of *SpreadChange* variance
AGA	77.6	501.5	0.0018	3.82×10^{-7}	6.41×10^{-4}	3.56×10^{-5}
CGA	102.5	1879.9	0.0022	1.96×10^{-7}	5.25×10^{-4}	7.44×10^{-5}
LGA	43.7	1035.7	0.0016	2.08×10^{-7}	6.78×10^{-4}	6.31×10^{-5}
CLPS-GA	36.4	355.5	0.0020	4.94×10^{-8}	7.38×10^{-4}	2.61×10^{-5}

75 % or higher, CLPS-GA requires the fewest iterations to reach its convergence criteria, and the average number of iterations taken to converge is only 36.4, much fewer than those in other algorithms. As indicated by its lowest variance of iteration numbers and mean of *SpreadChange* variance at 355.5 and 2.61×10^{-5}, respectively, CLPS-GA achieves the best stability among these algorithms. However, this happens only when cases of high similarity exist, and if no case is introduced, then CLPS-GA just degenerates into regular GA with MPCO. Careful scrutiny on data in Tables 7.4 and 7.5 reveals that under this circumstance of no similar cases, the CLPS-GA algorithm's convergence rate is lower than that of LGA and its stability is poorer than AGA. The effectiveness of CLPS-GA thus depends on the availability of highly similar cases.

LGA, by strengthening its local search in neighboring areas of optimal individuals, achieves slightly inferior performance in convergence rate compared to CLPS-GA, which can be observed from its average iterations 43.7 in Table 7.5, but has to compromise its stability due to its large variances either on iterations or *SpreadChange* through generations. CGA, by generating a Logistic sequence to help search in a larger range, obtains better result than CLPS-GA in the diversity of solutions with average distance 0.0022 versus 0.0020, but it has the worst performance in terms of stability and convergence rate, i.e., it has an exceptionally large value on mean and variance of iterations, 102.5 and 1879.9, respectively. Lastly AGA, by adjusting the crossover rate and mutation rate according to the fitness of individuals involved, neither stands out nor falls behind with its medium performance almost in every aspect.

As shown in Fig. 7.11, none of the Pareto fronts drawn is evidently close to the axis, which indicates the negligible difference in the quality of solutions obtained by each algorithm. Nevertheless, it can be easily seen from the left figure that solutions are distributed more evenly in CLPS-GA and CGA, and slightly unbalanced and concentrated in AGA. In examining the curves on the right figure, it can be easily distinguished that CLPS-GA and LGA take relatively fewer iterations than AGA and CGA to converge. Note that CGA has the highest diversity. Therefore, it's safe to say that diversity and efficiency go against each other and no algorithms can have it both ways.

In summary, each algorithm has their strengths and weaknesses, and CLPS-GA does best in terms of convergence rate and stability, and ranks only second to CGA

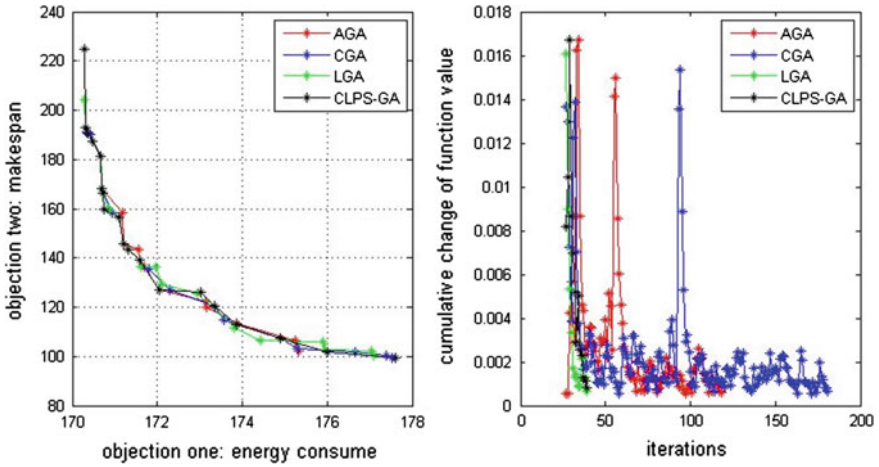


Fig. 7.11 Comparison of CLPS-GA and other enhanced GAs

in the diversity of solutions. Since the idea of introducing a multi-parent crossover operator and a case library is not contradictory to the strategies employed in other algorithms, there is still room for exploring other combinations of various strategies in order to improve further.

7.6 Summary

Service scheduling has always been a core component in cloud manufacturing system. However, previous studies on its model building and scheduling algorithms are either insufficient or far from satisfactory. Taken the advantage of population based configuration, we presented a new improved genetic algorithm comprised of Pareto searching operators and case library mechanisms. In summary, this chapter mainly includes the following contents.

(1) For addressing the OSCR problem, energy consumption and makespan are chosen as two objectives. The energy consumption model is formulated and simplified to adapt to network whose load information is unavailable. Meanwhile, imbalanced load distribution is considered to represent risk on the makespan and used as an effective strategy both to shorten the makespan and to realize load balance.
(2) Different from past works, which often convert a MOO into a SOO, diverse solutions distributed on the Pareto front are provided for decision-makers to select from. This helps meet various kinds of user's needs and make the service more considerate and universal.
(3) The proposed improved approach for Pareto solutions (CLPS-GA) is innovative and composed of a multi-parent crossover operator newly redesigned,

a two-stage algorithm structure, a case library, and a new concept of case similarity. Experimental results have shown its high performances in terms of convergence, stability and solutions' diversity in solving the subject MOO problem.

References

1. Tao F, Feng Y, Zhang L, Liao TW (2014) CLPS-GA: A case library and Pareto solution-based hybrid genetic algorithm for energy-aware cloud service scheduling. Applied Soft Computing 19:264–279
2. Wang J, Varman P, Xie C (2011) Optimizing storage performance in public cloud platforms. J Zhejiang Univ Sci C (Comput Electron) 12(12):951–964
3. Rivoire S, Shah MA, Ranganathan P, Kozyrakis C (2007) Joulesort: a balanced energy-efficiency benchmark. In: Proceedings of the ACM SIGMOD, international conference on management of data, NY, USA pp. 365–376
4. Bianchini R, Rajamony R (2004) Power and energy management for server systems. Computer 37(11):68–74
5. Ullman JD (1975) NP-complete scheduling problems. J Comput Syst Sci 10(3):384–393
6. Yu J, Buyya R, Ramamohanarao K (2008) Workflow scheduling algorithms for grid computing. Metaheuristics for scheduling in distributed computing environments, Springer, Heidelberg, pp 173–214
7. Armbrust M, Fox A, Grifth R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2009) Above the clouds: a berkeley view of cloud computing. Technical report, University of California at Berkeley
8. Buyya R, Pandey S, Vecchiola C (2009) Cloudbus toolkit for market-oriented cloud computing. In: Proceedings of the 1st international conference on cloud computing, Beiing, China, pp 24–44
9. Deelman E, Singh G, Su MH, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, Laity A, Jacob JC, Katz DS (2005) Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci Program J 13(3):219–237
10. Cao J, Jarvis SA, Saini S, Nudd GR (2003) Gridflow: workflow management for grid computing. In: Proceedings of the 3rd international symposium on cluster computing and the grid, Washington, DC, USA, pp 198–205
11. Furmento N, Lee W, Mayer A, Newhouse S, Darlington J (2002) Iceni: an open grid service architecture implemented with jinni. In: Proceedings of the ACM/IEEE conference on supercomputing
12. Amin K, von Laszewski G, Hategan M, Zaluzec NJ, Hampton S, Rossi A (2004) Gridant: a client-controllable grid workflow system. In: Proceedings of the 37th annual Hawaii international conference on system sciences, Big Island, HI, USA, pp 3293–3301
13. Taylor I, Wang I, Shields M, Majithia S (2005) Distributed computing with Triana on the grid. Concurrency and Comput Pract Experience 17(9):1197–1214
14. Ludascher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee EA, Tao J, Zhao Y (2006) Scientific workflow management and the kepler system. Concurrency Comput Pract Experience 18(10):1039–1065
15. Mayo RNP, Parthasarathy R (2005) Energy consumption in mobile devices: why future systems need requirements-aware energy scale-down. In: Proceedings of 3rd international workshop on power-aware computer systems, San Diego, CA, USA pp 26–40
16. Chase JS, Anderson DC, Thakar PN, Vahdat AM, Doyle RP (2001) Managing energy and server resources in hosting centers. Operating Syst Rev 35(5):103–116

17. Kephart JO, Chan H, Das R, Levine DW, Tesauro G, Rawson F, Lefurgy C (2007) Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In: Proceedings of 4th international conference on autonomic computing, Florida, USA, pp 1–10
18. Beloglazov A (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comput Syst 28(5):755–768
19. Srikantaiah S, Kansal A, Zhao F (2008) Energy aware consolidation for cloud computing. In: Proceedings of hotpower workshop on power aware computing and systems, San Diego, CA, USA
20. Beloglazov A, Buyya R, Lee YC, Zomaya A (2011) A taxonomy and survey of energy-efficient data centers and cloud computing systems. In: Advances in computers. Elsevier, Amsterdam, The Netherlands
21. Lei DM, Xiong HJ (2007) An efficient evolutionary algorithm for multi-objective stochastic job shop scheduling. In: Proceedings of international conference on machine learning and cybernetics, Hong Kong, China, pp 867–872
22. Jin Z, Yang Z, Ito T (2006) Metaheuristic algorithms for the multistage hybrid flow shop scheduling problem. Int J Prod Econ 100:322–334
23. Tang L, Liu W, Liu J (2005) A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment. J Intell Manuf 16:361–370
24. Ishibuchi H, Yamamoto N, Misaki S, Tanaka H (1994) Local search algorithms for flow shop scheduling with fuzzy due-dates. Int J Prod Econ 33:53–66
25. Pandey S, Wu L, Guru SM, Buyya R (2010) A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. Proceedings of the 24th IEEE international conference on advanced information networking and applications. Perth, WA, Austialia, pp 400–407
26. Niu SH, Ong SK, Nee AYC (2012) An enhanced ant colony optimiser for multi-attribute partner selection in virtual enterprises. Int J Prod Res 50(8):2286–2303
27. Li C, Li L (2007) Utility-based QoS optimization strategy for multi-criteria scheduling on the grid. J Parallel Distrib Comput 67:142–153
28. Knowles JD, Corne DW (2000) Approximating the non-dominated front using the pareto archived evolution strategy. Evol Comput 8(2):149–172
29. Coello Coello CA, Pulido GT (2001) A micro-genetic algorithm for multi objective optimization. In: Proceeding of the 1st international conference on evolutionary multi-criterion optimization, Zurich, Switzerland, pp 126–140
30. Coello Coello CA, Pulido GT, Lechuga MS (2004) Handling multiple objectives with particle swarm optimization. IEEE Trans Evol Comput 8(3):256–279
31. Mostaghim S, Teich J (2004) Covering Pareto-optimal fronts by sub-swarms in multi objective particle swarm optimization. Evol Comput 2:1404–1411
32. Nam D, Park CH (2000) Multi-objective simulated annealing: a comparative study to evolutionary algorithms. Int J Fuzzy Syst 2(2):87–97
33. Garcia-Martinez C, Cordon O, Herrera F (2007) A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. Eur J Oper Res 180(1):116–148
34. Chi-Keong G, Yew-Soon O, Chen TK (2009) In: Multi-objective memetic algorithms. Springer-Verlag, Berlin
35. Bingul Z (2007) Adaptive genetic algorithms applied to dynamic multiobjective problems. Appl Soft Comput 7(3):791–799
36. Vafaee F, Nelson PC (2009) Self-adaptation of genetic operator probabilities using differential evolution. In: Proceedings of the 3rd IEEE international conference on self-adaptive and self-organizing systems, San Francisco, US, pp 274–275
37. Qi RB, Qian F, Li SJ, Wang ZL (2006) Chaos genetic algorithm for multi objective optimization. In: Proceedings of the 6th congress on intelligent control and automation, pp 1563–1566

38. Gao MJ, Xu J, Tian JW, Wu H (2008) Path planning for mobile robot based on chaos genetic algorithm. In: Proceedings of the international conference on natural computation, pp 409–413
39. Ishibuchi H, Murata T (1998) A multi objective genetic local search algorithm and its application to flowshop scheduling. IEEE Trans Syst Man Cybern 28(3):392–403
40. Martinez CG, Lozano M, Molina D (2006) A local genetic algorithm for binary-coded problems. Parallel Probl Solving Nat 4193:192–201
41. Gen M, Cheng R (2000) Genetic algorithm and engineering optimization. Wily, New York
42. Engin O, Ceran G, Yilmaz MK (2010) An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. Appl Soft Comput 11(3):3056–3065
43. Netto MAS, Buyya R (2009) Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems. In: Proceedings of IEEE international symposium on parallel and distributed, pp 1–11
44. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82

Part IV
Application of Hybrid Intelligent Optimization Algorithms

Chapter 8
SFB-ACO for Submicron VLSI Routing Optimization with Timing Constraints

The arrival of submicron era has created a huge difference on VLSI (very large scale integration): delay on interconnects has far exceeded that on gates so the total delay for a sink can no longer be simply assessed by the length of weighted edges which makes its routing more complicated than ever. Traditional methods for VLSI routing are either infeasible or with a low precision. In this chapter multiple objectives are comprehensively reflected as a cost the optimization problem has been abstracted as constructing a minimal rectilinear Steiner tree with rectangular obstacles (MRSTRO) under timing constraints. Then the relationship between cost sink delay is cautiously discussed partially proved to be contradictory using Elmore delay model which is of high fidelity. To effectively address the MRSTRO problem a synergy feedback based ant colony algorithm (SFB-ACO) is configuredimplemented. In SFB-ACO a synergy function is designed to lead each branch to join others thus reducing the total tree length. Additionally according to the intrinsic contradiction between objective constraint a constraint-oriented feedback module is introduced with the purpose of preventing over-constrain while regulating the formation of solutions. With configuration principle two modules are uniformly connected with existing ACO operators to form a hybridization of deterministic strategies evolutionary process. The experimental results have verified the advantage of SFB-ACO compared to other algorithms or practices on VLSI global routing.

8.1 Introduction

Global routing [1–4] is to arrange each part of the net into different wiring channels and determine the connection of nodes and their initial wiring courses while satisfying certain design requirements. Its result could have a significant impact on the success of follow-up detailed routing and the overall performance of

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_8

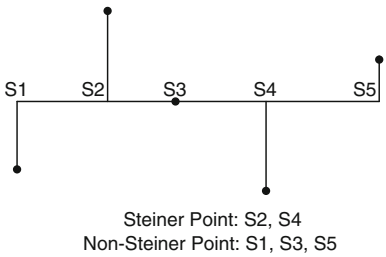


Fig. 8.1 Steiner points and non-Steiner points

the chip [5, 6]. Therefore, it is a core link in very large scale integration (VLSI) physical design. Wires in VLSI can be divided into several types: *signal line*, *power line*, *ground line*, *clock line*, etc., with various optimization objectives of each type. Generally speaking, the total length of interconnect, and the area of wiring district are required to be as small as possible, and time delay for signal lines, clock skew for clock lines are also needed to be considered. Other extra objectives include: power dissipation and heat loss should be reduced; noise and crosstalk between lines should be avoided, etc. These objectives can be comprehensively reflected by weighing each edge in the net; then the optimization problem is narrowed down on minimizing the weighted length of wires, which is usually called a cost. The procedure for constructing interconnects for VLSI global routing has been abstracted as a minimal rectilinear Steiner tree with rectangular obstacles (MRSTRO) problem.

Studies on Steiner tree in Graph theory can be traced to 1941, when Courant and Robbins [7] pointed out that for a net consisting of n endpoints, at most n-2 points are needed to be introduced, and together with the original points, the cost of Steiner tree established on them can be reduced to the lowest. These introduced points are called Steiner points. (Note that not all yielding points are Steiner points, as illustrated in Fig. 8.1) Apparently, how to pick them correctly is a key issue of Minimal Steiner tree (MST) problem [4, 8–10], which has been proved to be a NP-hard problem [4, 9, 11]. The computational amount of some exact algorithms [9, 10, 12, 13] for Steiner tree increases exponentially as the number of nodes increases. Particularly, wires in VLSI must follow Manhattan routing architecture [14, 15], in which only two perpendicular wiring directions are allowed. The optimal tree spanned under such architecture is formatted as a minimal Rectilinear Steiner tree (MRST). Algorithms to perform MRST construction are usually computational-expensive in space, since it requires divisions between each pair of nodes and separated considerations on weighted cost of divided sections, thus a memory usage of prohibitively huge size.

To overcome these shortcomings, Hanan [16] defined Hanan points and gave out a well-known theorem that for any endpoints collection P, there exists a minimum Steiner tree solution, whose Steiner points set S is a subset of its Hanan points set U. Later Snyder [13] demonstrated that the above theorem can be extended to Manhattan space with higher dimensions. Such endeavors greatly make MRST a less overwhelming task. In addition, attentive scholars found that

MST can serve as a good estimation to MRST, and tried to get an approximated solution by using MST as a starting point in efforts to either decompose nets into several two-pin subnets to ease maze routing [3, 17, 18], or simply adopt a pattern technique to restrict the connecting to be L-shape or Z-shape [3, 19, 20]. Hwang [21], in 1976, stated and proved the formula that $cost(MST)/cost(MRST) \leq 3/2$, and this number cannot be improved, which means the approximation cost of MST and MRST will reach 3/2 in the worst cases. Such a poor precision cannot be accepted in VLSI routing design. Meanwhile, the existence of obstacles, which is created by various macro modules, IP modules and some others on the chip, results in not only a more complicated process for deriving MRST from MST, but also a larger discrepancy between their costs. All above have led to an increasing popularity in finding MRST in a more straight and accurate way.

Thus far, numerous mature algorithms directed at MRST have been put forward, such as Geo Steiner package [12, 22], edge based heuristic algorithm [23], 1-Steiner heuristic [24–26] etc. Recently, FLUTE [27, 28] propositioned with improved performance for it is optimal for nets up to degree 9 and is still very accurate for nets up to degree 100. It has been well appreciated and its direct applications are BoxRouter [3, 29] and FastRouter [3, 30, 31]. For tackling MRSTRO problems [32–36] constructed an obstacle-avoiding Steiner tree for an arbitrary λ-geometry by Delaunay triangulation, and demonstrated that it outperformed the conventional construction-by-correction approach [35]. Most algorithms mentioned above are heuristic, which facilitate the finding of a near-optimal solution within a relatively short period of time, thereby has been widely used in multicast network optimization [4]. However, most nets in VLSI circuits have a low degree [28], so rather than having a low runtime complexity, the quality of solution is a more important factor. Discouragingly, up till now, none of the heuristic algorithms can attain twice better performance in the best cases than in the worst ones. Rita and Bryant successfully applied genetic algorithm (GA) in MRSTRO based on MST [36], and Consoli [37] proposed a Jumping Particle Swarm Optimization methodology for addressing the minimum labelling Steiner tree problem, both of which imply a bright and prospective application of intelligent algorithms in VLSI routing [38].

Among intelligent algorithms, ant colony optimization (ACO) [39–41] is a kind of bionic algorithm suggested and quickly developed by Dorigo. Using pheromone to transmit messages between ants, its biggest characteristic is to subtly integrate information of historical experience, excellent solutions and their interactions in a distributed way through weighted edges in the searching space. By receiving positive feedback of pheromone as well as heuristic guidance, the searching and message exchanging efficiency and its quality can be guaranteed, thus gradually becoming a very promising algorithm. At present, ACO is still at the very outset of its development, and is mainly used in path planning and has received better results than genetic algorithm (GA) and simulated annealing algorithm (SA). MRSTRO belongs to path planning, whose optimal route can be excavated relying on distributed information of edges. However, it distinguishes itself from general

path planning for it usually contains multiple endpoints. Researches on how to apply ACO on multi-terminal connection is still rare.

On the other hand, integrated circuit develops towards a high-speed and high-integration-level trend. With the coming of deep submicron times, interconnect on VLSI has become thinner and longer, leading to a substantial increase on its resistance and capacitance. Consequently, the delay on interconnect is no longer negligible, while that on gate decreases as its feature size shrinks. For instance, for 100 nm, the intrinsic switching delay of a MOSFET is 5 ps, whereas the RC response time for 1 mm of interconnect is 30 ps. At 35 nm, this 6-to-1 differential turns into a 100-to-1 difference [42]. These changes have made interconnect routing on VLSI very different from before [43]. Previous Linear delay estimator being the Manhattan distance between nodes, which is more commonsense-based, pales in fidelity compared to Elmore delay [42, 44–47], in which a routing tree with shortest length, though possessing a comparative small wiring area, and sometimes a relatively better synchronicity in critical sinks, pays at other prices. By maximizing sharing of tree's branches, it adds extra nodes to the mainstream from source to sinks and this could severely lengthen the delay. As a result, the delay constraints at some sinks may be violated, which adversely affects the performance of circuits or even leaving it malfunctioning. Since the calculation of each sink's delay depends on the structure of Steiner tree and is highly coupled with other branches, such information is rather difficult to be incorporated into distributed edges and hence cannot be appraised by tree's cost. Therefore, traditional approach, to empirically identify delay as connecting length, and then focus objectives of global routing on reducing the total cost, is not applicable in today's deep submicron regime.

Based on the above analysis, the delay on each sink is intrinsically contradicted to the total wiring length. In other words, to ensure the least delay on a particular sink, what we need to do is just to link it directly with the source, which may lead to a star-like topology of Steiner tree. Obviously, its total cost is relatively high and thus unwanted. When the delay calculated from the resulting tree is less than the given constraint, it usually means that there is still possibility for merging branches to reduce cost. Therefore, the ideal situation is that delay on each sink should be less than but as close as possible to their respective constraints. Normal methods to deal with optimization problems with constraints can be roughly summarized as follows: one is to accept or abandon a solution (AAS) directly in relation to its eligibility to meet the constraints, and the other one is to bring in a penalty function to turn the questions into non-constraint ones. The former one fails to make full use of solutions with good target values but cannot satisfy the constraints, and in the latter one, likewise, such solutions suffer from punishment and then degrade. In this context, we hope to take advantage of delay information in the last iteration as guidance for generating solutions in the next. Inspired by the positive feedback in ACO, and considering the contradictory relationship between objectives and constraints in VLSI global routing, a negative feedback is introduced to reconcile merging of branches according to the degree how a constraint is satisfied. Specifically, if the delay constraint on a sink is severely violated, its

synergy coefficient decreases so as to restrain meeting with others, and otherwise increases to encourage so.

The primary works of this chapter are as follows:

Optimization on global routing in VLSI is abstracted as a MRSTRO problem. For addressing this problem, the MST constructing process is skipped, and an enhanced ACO, which contains a synergy function other than the pheromone and heuristic factors, is proposed and applied in multi-terminal path planning problems.

Differences on VLSI routing between today's deep submicron era and before are carefully investigated. A more accurate Elmore delay is employed and a constraint-oriented feedback is introduced to adjust branch's merging with others to prevent the case of over-constrain.

Through experimental tests, the effectiveness of synergy function and constraint-oriented feedback in our proposed SFB-ACO is verified by comparisons with other algorithms or practices.

8.2 Preliminary

8.2.1 Terminology in Steiner Tree

The model for VLSI global routing in this chapter is based on MRSTRO, where Steiner tree consists of a collection of given points, additional introduced Steiner points and their connecting relationships. For any two points in Steiner tree, one and only one path can be found. Other requirements in VLSI routing include: either horizontal or vertical wiring lines, no transverses across functional area on the chip. Here, some interpretations related to Steiner tree [8, 10] need to be given as follows.

- ***Root, Node, Leaf and Edge***
  Any point consisting of a Steiner tree is called a node, whose set, denoted as T, is a union of P and S. Connection between two nodes is called an edge, which defines a parent-child relationship. Within the structural hierarchy in the tree, there is a node with special status, usually called as a root. The closer to the root, the higher rank of the node is. Leaf is defined as a node whose degree is 1, namely the point that only has one connection.
- ***Steiner Points and Hanan Points***
  Any additionally introduced points that can help reduce the length of the spanning tree are called as Steiner Points. By drawing horizontal and vertical lines through points in P, we can obtain a Hanan grid. The intersections of the grid are called Hanan points, and its collection is indicated as U.
- ***Mainstream, Segment and Subtree***
  For any element in P, the path from it to the root is called its mainstream, and the connection between two adjacent nodes in the mainstream is called a

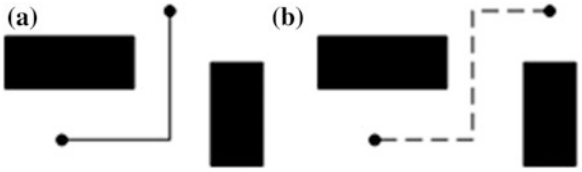


Fig. 8.2 Instances of connectivity **a** connectable **b** non-connectable

segment. A partial tree rooted in node T_i, and consisting of T_i and all its child nodes are called a subtree of T_i, denoted as $Sub(i)$.

- ***Connectivity***
  In the wiring diagram, the existence of obstacles may prevent some Hanan points to be selected as Steiner points. In the cases where exists a pair of nodes P_1 and P_2, and their Hanan points U_1 and U_2, their connection cannot be completed by simply choosing U_1 or U_2 to be the yielding point, but requires two or more, such situation is called non-connectable; otherwise, we call it connectable, illustrated in Fig. 8.2.

8.2.2 Elmore Delay

Elmore delay is a relatively accurate and commonly used model when calculating signal delay over the network. In today's deep submicron era, delay on the VLSI interconnect can no longer be ignored. For a wire with length L, it can be divided into N segments and each is measured as ΔL; then it can be described by a RC network model illustrated in Fig. 8.3.

Assuming that the wire itself is homogeneous, that is, the resistance and capacitance per unit length is a constant, denoted as R_{rate} and C_{rate}, respectively, then the total delay along this wire can be expressed in Eq. (8.1) [45].

$$\begin{aligned}\tau_L &= (R_{rate}\cdot\Delta L)(C_{rate}\cdot\Delta L)+2(R_{rate}\cdot\Delta L)(C_{rate}\cdot\Delta L)+\cdots+N(R_{rate}\cdot\Delta L)(C_{rate}\cdot\Delta L)\\ &= R_{rate}\cdot C_{rate}\cdot(\Delta L)^2\cdot\sum_{i=1}^{N} i=\frac{1}{2}R_L\cdot C_L\end{aligned} \tag{8.1}$$

where, R_L and C_L represent the wire's total resistance and capacitance.

For a node T_i in Steiner tree, the signal delay from the root T_0 to T_i can be formulated as Eq. (8.2) [46].

$$\tau(T_i) = R_{T_0}\cdot C_{T_0} + \sum_{all\ segments\ along\ mainstream(i)} \tau_{e_j} \tag{8.2}$$

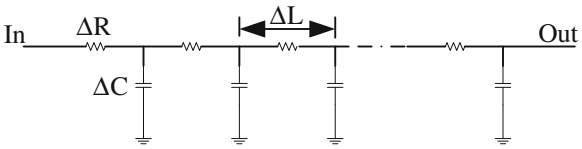


Fig. 8.3 RC model for a wire

$$\tau_{e_j} = R_{T_j} \cdot C_{T_j} + R_{e_j}\left(\frac{C_{e_j}}{2} + C_{sub(j)}\right) \tag{8.3}$$

$$C_{sub(j)} = \sum_{all\ nodes\ in\ subtree(j)} C_{T_k} + \sum_{all\ edges\ along\ subtree(j)} C_{e_l} \tag{8.4}$$

In Eq. (8.2), $R_{T_i}(i = 0, 1, 2, \ldots)$ represents the resistance to drive node T_i, $C_{T_i}(i = 0, 1, 2, \ldots)$ represents capacitance of node T_i, and e_j represents the edge from T_j to its next nearest node along T_i's mainstream. Correspondingly, delay along this edge is denoted as τ_{e_j}, and computed by Eq. (8.3), where R_{e_j} and C_{e_j} respectively represent the total resistance and capacitance of the edge, and $C_{sub(j)}$ is defined as the equivalent capacitance of subtree rooted in T_j's nearest child node along the mainstream, which is the sum of capacitance of all nodes and edges in the subtree. If the child node of T_j is a leaf, then the value of $C_{sub(j)}$ is identical to the capacitance of this leaf.

Up till now, for each node in the Steiner tree, its delay from the root along its mainstream can be calculated iteratively according to Eqs. (8.2)–(8.4).

8.2.3 Problem Formulation

This section discusses MRETRO problem with timing constraints for VLSI global routing in a deep submicron era. In this section, the relationship between the sink delay and tree length is scrutinized, and appropriate candidate pool for Steiner points is determined on account of solution precision and space complexity.

Given a point set $P = \{P_i | i = 1 : n\}$ corresponding to the sinks on the VLSI chip to be optimized, where n is the number of sinks, and root T_0 corresponds to the source on chip. For each sink and source, it can be located by its coordinate (x_i, y_i) on board. Besides, there is a collection of modules, viewed as obstacles, the actual shape of which does not affect the area for wiring because of the Manhattan rule in VLSI, and thus can be formulated or divided into a number of rectangles. These rectangular obstacles are denoted as $R = \{R_i | i = 1 : r\}$, where r is the number of obstacles, and its position and size are expressed by its bottom-left and upper-right vertex coordinates. Upon the basis of sinks and source, some other special points, known as Steiner Points, are needed to be introduced. How to

construct a MRST using these points, and at the same time, not violate the delay constraint of each sink? This is an issue which needs to be addressed here.

As mentioned before, the wiring length, chip area, power consumption, heat loss, and clock synchronicity can be accessed by the total cost of weighted edges in the spanning tree. Here all the wires in the chip are assumed to be homogeneous, which means that all edges are of uniform weight (which is set to 1), so that minimizing the total cost of Steiner tree is equivalent to minimizing its total length. Also, the delay constraint of each sink is set to be $T_{limit} = \{T_{limit}(i)|i = 1 : n\}$, then the optimization problem can be formulated as follows.

$$Min\left(\sum_{all\ segments} L_e\right) \tag{8.5}$$

$$subject \quad to \quad \tau(T_i) \leq T_{limit}(i), \quad \forall i = 1 : n \tag{8.6}$$

In Eqs. (8.5) and (8.6), L_e represents the length of each segment in the spanning tree, and τ and T_{limit} respectively represent the actual Elmore delay of the sink and its delay constraint.

Apparently, the selection of Steiner points is the key to solve above problem. Note that if rendering the candidate pool to be infinite or with little limitation, it will unavoidably increase the space complexity of the problem.

Theorem 1 (Hanan [16]) *For any MRST, all of its Steiner points are Hanan points.*

Corollary 1 *If a MRSTRO problem is solvable, its optimal solution can be obtained by selecting Steiner points from Hanan points set or from points located in the rim of obstacles.*

Corollary 2 *For two points T_i and T_j to be connected, the shortest path between them is equal to the Manhattan distance between them, as defined in Eq.* (8.7)*, when they are connectable; otherwise it should at least contain one portion of obstacle's edge.*

$$D(T_i, T_j) = \left|x_i - x_j\right| + \left|y_i - y_j\right| \tag{8.7}$$

Theorem 2 *For a partial tree T and an unconnected point P_k outside the tree, the best location in segment for a Steiner point to connect P_k to T to control the tradeoff between tree length and sink delay, should lie between SP and CUC, as shown in Fig.* 8.4*, where SP is the shadow point of P_k to the segment, and CUC is the closest upstream connection to P_k.*

Proof Let T_i denote the closer-to-root endpoint of the segment to be connected, which is CUC, and its coordinate to be (0, 0). Let the other endpoint, i.e., closest downstream connection (CDC), to be indicated by T_j, with its coordinate being (x_j ,0). Let the coordinate of P_k to be (x_k, y), and its shadow point in segment to be $(x_k, 0)$. Let R_{T_i} and C_{T_i} denote the resistance and capacitance of node T_i, respectively. Let R_s and C_s respectively represent the Steiner point's resistance and capacitance if it does not lie on CUC or CDC. Denote the resistance and

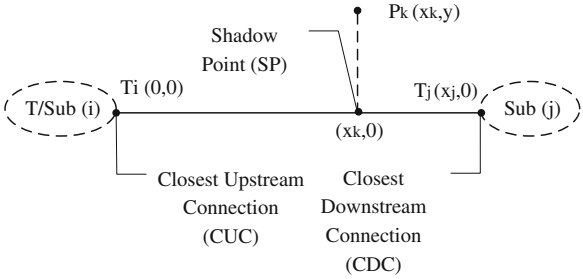


Fig. 8.4 Diagram of CUC, SP and CDC

capacitance per unit length on segment by R_{rate} and C_{rate}, respectively, and the equivalent capacitance of subtree rooted in T_j to be $C_{sub(j)}$, and assume that the coordinate for selected Steiner point on segment is $(x, 0)$. Then according to Elmore model, delay from source to node P_k along its mainstream can be expressed as follows.

$$\tau(P_k) = \tau(predecessor) + \tau_{e_i} + \tau_{e_s} \tag{8.8}$$

In Eq. (8.8), $\tau(predecessor)$ represents the signal delay from source to node T_i, τ_{e_i} represents delay from node T_i to the selected Steiner point, and τ_{e_s} represents delay from the Steiner point to node P_k.

Additionally, we can easily infer from Eqs. (8.2)–(8.4) that the connecting of P_k only affects the value of C_{sub}, and for each $C_{sub(i)}$ in the upstream route, it can be expressed as in Eq. (8.9).

$$C_{sub(l)} = C_{predecessor} + C_{sub(i)}, \quad \forall T_l \in Predecessor(T_i) \tag{8.9}$$

So that we can rewrite Elmore delay to be a linear function of $C_{sub(i)}$, expressed as follows.

$$\tau(predecessor) = ConA + ConB \cdot C_{sub(i)} \tag{8.10}$$

Where $ConA$ and $ConB$ are both constant, which are only related to the resistance and capacitance of T_i's upstream route, respectively. The change of $C_{sub(i)}$ has nothing to do with the value of $ConA$ or $ConB$.

Hence the influence of Steiner point's location on $C_{sub(i)}$ can be calculated according to Eq. (8.11).

$$C_{sub(i)} = C_{T_i} + C_S + C_{P_k} + C_{sub(j)} + C_{rate} \cdot \left(x_j + |x_k - x| + y\right) \tag{8.11}$$

Also, its influence on τ_{e_i} and τ_{e_s} can be expressed as in Eqs. (8.12) and (8.13), respectively.

$$\tau_{e_i} = R_{T_i} \cdot C_{T_i} + R_{rate} \cdot x \cdot \left\{ C_{rate} \cdot \frac{x}{2} + C_S + C_{sub(j)} + C_{P_k} + C_{rate}\left[(x_j - x) + |x_k - x| + y\right] \right\} \quad (8.12)$$

$$\tau_{e_S} = R_S \cdot C_S + R_{rate} \cdot (|x_k - x| + y) \cdot \left(C_{rate} \cdot \frac{|x_k - x| + y}{2} + C_{P_k} \right) \quad (8.13)$$

According to Eqs. (8.8)–(8.13), we can easily find that $\tau(P_k)$ is a piecewise quadratic function of x, expressed as follows.

$$\tau(P_k) = A + Bx + Cx^2 \quad (8.14)$$

where

$$A = \begin{cases} ConA + ConB\left[C_{T_i} + C_S + C_{P_k} + C_{sub(j)} + C_{rate}(x_j + x_k + y)\right] + R_{T_i}C_{T_i} \\ \qquad + R_S C_S + R_{rate}(y + x_k)\left(C_{rate} \cdot \dfrac{y + x_k}{2} + C_{P_k}\right), \quad 0 \le x \le x_k \\ ConA + ConB\left[C_{T_i} + C_S + C_{P_k} + C_{sub(j)} + C_{rate}(x_j - x_k + y)\right] + R_{T_i}C_{T_i} \\ \qquad + R_S C_S + R_{rate}(y - x_k)\left(C_{rate} \cdot \dfrac{y - x_k}{2} + C_{P_k}\right), \quad x_k < x < x_j \end{cases} \quad (8.15)$$

$$B = \begin{cases} -ConB \cdot C_{rate} + R_{rate}\left(C_S + C_{sub(j)} + C_{rate} \cdot x_j\right), & 0 \le x \le x_k \\ ConB \cdot C_{rate} + R_{rate}\left(C_S + C_{sub(j)} + C_{rate} \cdot x_j\right), & x_k < x < x_j \end{cases} \quad (8.16)$$

$$C = \begin{cases} -R_{rate} \cdot C_{rate}, & 0 \le x \le x_k \\ R_{rate} \cdot C_{rate}, & x_k < x < x_j \end{cases} \quad (8.17)$$

Other nodes, apart from those which directly connect to the source without any other nodes within the mainstream, their signal delay will also be affected due to the newly connected point P_k. Among them, nodes located on $T/Sub(i)$ mainly suffer from the change of $C_{sub(i)}$, and the major cause for delay variation of those on $Sub(j)$ will be the increase of segment number and their corresponding delay change along the mainstream. Delay on these nodes is also a piecewise quadratic function of x, which can be get as above.

Qualitatively drawing curves to depict signal delay of P_k and nodes distributed in other positions, as shown in Fig. 8.5, we can easily tell that delay on all sinks increase when the Steiner point is inserted after *SP*. And apparently, the tree length is longer compared to the situation when the Steiner point lies before *SP*. Therefore, an appropriate location for the new Steiner point should be between *CUC* and *SP*. In this region, the length of spanning tree gradually decreases when slowly shifting Steiner point backwards, and reaches its lowest point when at *SP*. Another conclusion drawn from Fig. 8.5 is that, delay on each sink continuously changes as the Steiner point moves between *CUC* and *SP*, and some of them

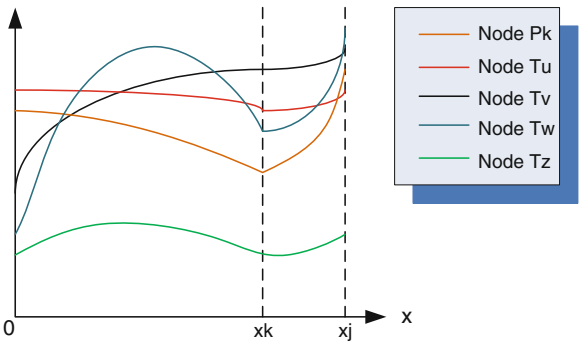


Fig. 8.5 Change of signal delay on different nodes

change in the opposite direction, which implies, that the timing conditions at different sinks are sometimes contradictory when adjusting position of one Steiner point. This makes it possible for us to artificially regulating the synergy function of branches in order to meet their respective timing constraints.

From above, we also see that limiting the candidate Steiner points to the Hanan pool is actually not conductive to the adjustment of sink's delay. And according to Corollary 1 and 2, the Hanan points are not enough if the design model is non-connectable itself. Taking the space complexity into account, $S = U_{refined} \cup RIM \cup PEAK$ can serve as an appropriate candidate pool for Steiner points, where $U_{refined}$ is a collection consisting of Hanan points that lies off the obstacle region, *RIM* is comprised of intersections created by drawing horizontal and vertical lines from sinks to the obstacles' rims, and *PEAK* represents the collection of all rectangular obstacles' vertices, as illustrated in the right panel of Fig. 8.8. □

8.3 SFB-ACO for Addressing MSTRO Problem

This section starts with a brief overview of ACO on two-endpoint path planning, and based on it, a SFB-ACO algorithm encompassing a synergy function and constraint-oriented feedback is proposed and then applied on multi-terminal routing optimization presented before.

8.3.1 ACO for Path Planning with Two Endpoints

The basic idea for traditional ACO can be summarized as below. Using ants' paths to represent feasible solutions, all paths searched can constitute the whole solution space for the given optimization problem. Let ants release more pheromone on the path whose total length is relatively shorter, and as time goes by, more and more pheromone can be accumulated on such paths, and thus they are more likely to be selected by other ants. Influenced by such an intense positive feedback, ants will

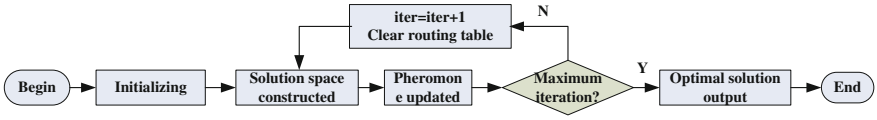


Fig. 8.6 Framework of standard ACO

ultimately converge into an optimal path with shortest length, and this path is also called as the optimal solution.

The main framework for standard ACO on path planning with two-endpoints is depicted in Fig. 8.6.

At first, pheromone concentration on all edges is the same, denoted as $\zeta_{ij}(0) = \tau_0$. Ant k ($k = 1,2,\ldots,m$) will choose the next node to visit according to the amount of pheromone deposited on edge as well as heuristic information, and the corresponding transferring rate for ant k to move from node i to node j can be denoted as $P_{ij}^k(t)$, and calculated in Eq. (8.18).

$$P_{ij}^k = \begin{cases} \dfrac{(\zeta_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum\limits_{S \in allow_k} (\zeta_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}, & S \in allow_k \\ 0, & S \notin allow_k \end{cases} \tag{8.18}$$

where ζ_{ij} represents the pheromone concentration on edge between i and j, η_{ij} represents heuristic function to signify expectation for ants to move from i to j, $allow_k$ represents the node collection that are allowed for ant k to visit, α is a pheromone factor, whose value represents the importance degree of pheromone concentration in ant's transferring and value of β, referred to as heuristic factor, represents that degree of heuristic information.

At the same time, the pheromone concentration on each edge will be updated with its formulation expressed as below.

$$\zeta_{ij}(t+1) = (1-\rho) \cdot \zeta_{ij}(t) + \Delta\zeta_{ij} \tag{8.19}$$

$$\Delta\zeta_{ij} = \sum_{k=1}^{n} \Delta\zeta_{ij}^k \tag{8.20}$$

where ρ represents the degree of pheromone evaporation, and its value lies on region [0,1], $\Delta\zeta_{ij}^k$ represents pheromone released by ant k on edge connected from i to j, and $\Delta\zeta_{ij}$ represents the total pheromone released by all ants on this edge.

As for the updating mechanism, Dorigo has given three different models, which are ant cycle system, ant quantity system, and ant density system. Among them, the first one employs the global information on ants' routing, thus being most commonly used. Its updating mechanism is introduced as follows.

$$\Delta\zeta_{ij}^{k} = \begin{cases} \frac{Q}{L_k}, & if\ ant(k)\ visit\ node(j)\ from\ node(i) \\ 0, & otherwise \end{cases} \tag{8.21}$$

where Q is a constant representing the total amount of pheromone released in one cycle, and L_k is identical to the length of ant k's route.

Standard ACO is perfectly suitable for shortest path planning with a sole source and destination. However, in a VLSI circuit board, there are multiple sinks and one source; what requires to be optimized is not the separate path from each sink to the source as in standard ACO, but the whole spanning tree created by all these points. In other words, no branch is completely independent, and only by merging branches in the maximum degree can we expect the shortest length of the tree. That's the reason that a synergy function is introduced in our proposed SFB-ACO.

8.3.2 Procedure for Constructing Steiner Tree Using SFB-ACO

Here we incorporate a synergy matrix γ with size $n \times n$, where $\gamma(i,j)$ represents the function for branch i to join in branch j. The procedure for constructing a Steiner tree is described as in Fig. 8.7. For n sinks to be connected in VLSI, let the number of ants in one group to be n, and that of ant groups to be m. Let the initial positions for n ants in one group to be the places where sink 1, sink 2, …, sink n lie. S is the candidate pool for Steiner points, and in combination with the source and sinks, they make up a point collection for ants to visit. Similar to standard ACO, each ant chooses its next node according to the transferring rate, and creates its own routing table. If any ant in the group transfers to the source or to nodes that have previously been visited by the other ants in its group, it succeeds in finishing its task and its travelling ends. Upon all ants in the group finish their tasks, we check the route to see whether it is a Steiner tree and record its length if yes. Otherwise, if any ant encounters a dead corner, i.e., there are not any allowed nodes to choose, we label a failure on the group and cancel all movements of its ants. Hereto, we call it one cycle. The pheromone concentration is updated once in a cycle, and only the group who successfully finishes the task can release pheromone on its path. As the pheromone accumulates through several cycles, the optimal Steiner tree can be found.

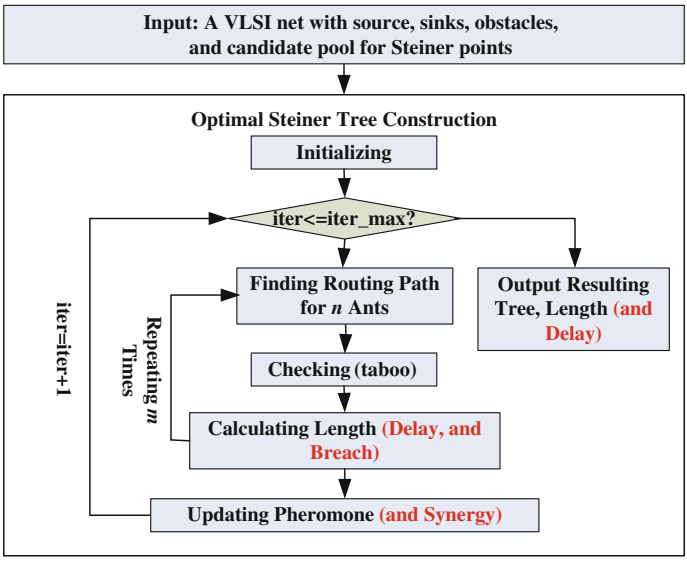


Fig. 8.7 Framework of SFB-ACO

The pseudo code for function *FindRoute* in Fig. 8.7 is as follows. In line 9, the transferring rate is calculated as follows.

```
Function taboo = FindRoute(source, sinks, obstacles, candidate Steiner points)
Step1   Initialization taboo, allowedmember
Step2   While (allowedmember ~= NULL)
Step3       Find nodes nearest to the taboo(end) from four directions
Step4       If (edge between node i and taboo(end) goes across any obstacle)
Step5           Let node = node / i
Step6       End if
Step7       Let node = node / taboo
Step8       If (node ~= NULL)
Step9           Calculate transferring rate
Step10          Select node and update taboo
Step11          Calculate transferring rate
Step12          If (node == root || node is a member of other ant's taboo)
Step13              Let current member be erased from allowedmember
Step14          End if
Step15      Else
Step16          Record failure on this round and break
Step17      End if
Step18  End while
Step19  Output the routing table of n ants
```

$$P_{ij}^{k} = \begin{cases} \dfrac{(\varsigma_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta} \cdot fun_{\gamma_{ij}}^{k}}{\sum_{s \in allow_k} (\varsigma_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta} \cdot fun_{\gamma_{ij}}^{k}}, & S \in allow_k \\ 0, & S \in allow_k \end{cases} \tag{8.22}$$

where

$$\eta_{ij} = 1/D\left(T_0, T_j\right) \tag{8.23}$$

$$fun_{\gamma_{ij}}^{k} = \prod_{r=1, r\neq k}^{n} \left[\frac{\sum_{l=1}^{N_{node}^{r}} \frac{1}{D\left(T_j, T_{tabu^r(l)}\right)}}{N_{node}^{r}} \right]^{\gamma_{kr}} \tag{8.24}$$

In Eq. (8.22), η_{ij} is identical to the Manhattan distance from source to node j, expressed as in Eq. (8.23). $fun_{\gamma_{ij}}^{k}$ represents the synergy function for ant k to transfer from node i to node j, whose value can be obtained by Eq. (8.24), where $T_{tabu^r(l)}$ represents the node in the route table, N_{node}^{r} represents the number of nodes that have been visited by ant r, and γ_{kr} represents the importance of synergy for branch k to join in branch r.

The pseudo code for function *Checking* in Fig. 8.7 is as follows.

```
Function flag = Checking(taboo)
Step1    Initialize flag to be 1
Step2    If (root is not a member of taboo)
Step3        Let flag = 0
Step4    Else
Step5        For i = 1 to n
Step6            Let path(i) record the path sink i which can go as far as it can toward the root
Step7            If (root is not the end of path(i))
Step8                Let flag = 0 and break
Step9            Else
Step10               If path(i) contains repeating nodes
Step11                   Let flag = 0 and break
Step12               End if
Step13           End if
Step14       End for
Step15   End if
Step16   Output checking result indicated as flag
```

Another innovative practice referring to the dealing with constraints will be presented in the next subsection.

8.3.3 Constraint-Oriented Feedback in SFB-ACO

Directed by the above analysis, Elmore delay on each sink is closely related to the number of Steiner points, their positions along the mainstream, and the connecting topology of other branches. To put it simple, the more meeting with others, the

more reduction it may cause on the length of spanning tree, whereas adding delay on relevant sinks. Therefore, the value in the synergy matrix will have direct impact on its corresponding sink's delay. Different from early methods dealing with solutions that break the constraints, which is either abandoning them or punishing them, this chapter presents a constraint-oriented feedback on elements in γ with the purpose to prevent the case of over-constrain.

First, the definition of constraint breach should be clarified.

$$Breach(T_i) = \tau(T_i) - T_{limit}(i) \tag{8.25}$$

Obviously, any positive value in vector *Breach* reveals a violation to the constraint. Otherwise, it may indicate a situation of over-constrain, which means that there is still space for further reducing the tree's length. Therefore, the ideal value in vector *Breach* should be equal to zero. However, this may be rather difficult, since the candidate pool for Steiner points we select is far from infinite. For that reason, the value in *Breach* should be lesser and as close as possible to zero.

The mechanism to regulate γ based on *Breach* is shown in Fig. 8.7 with red marks, in which the procedure of finding route path for n ants is the same as in Sect. 8.3.2, and the calculation of sink delay is based on Elmore model given in Sect. 8.2.

In addition, the pseudo code for pheromone updating function *UpdatePh* is as follows.

```
Function zera = UpdatePh(zeta)
Step1    Initialize Delta_zeta, rho
Step2    For i = 1 to m
Step3        If (vector{delay_i − 1.5×T_limit_i} contains no positive numbers)
Step4            Calculate decay_zeta
Step5            For each edge in the routing path
Step6                Let corresponding element in Delta_zeta increase by Q/Length(i)× decay_zeta
Step7            End for
Step8        End if
Step9    End for
Step10   Let zeta = (1 - rho)× zeta + Delta_zeta
Step11   Output resulting matrix zeta
```

Coefficient *decay_zeta* in above pseudo code is calculated as in Eq. (8.26).

$$decay\_zeta = e^{-\lambda \times \left(\sum_{l=1:n, \text{ and } \text{Breach}_i(l) > 0} |Breach_i(l)| \right)} \tag{8.26}$$

where λ is a constant to be determined, and *decay_zeta* represents the decaying degree of pheromone accumulated because of violation of constraints.

The pseudo code for synergy regulation function *RegulateSy* is as follows.

Function *gama* = RegulateSy(*gama*)
Step1 Initialize p1, p2, p3, p4
Step2 For $i = 1$ to m
Step3 For $j = 1$ to n
Step4 If $Breach_i(j) > 0$
Step5 Multiply the jth row of *gama* by $e^{-p1 \times Breach_i(j)}$
Step6 Multiply the jth column of *gama* by $e^{-p2 \times Breach_i(j)}$
Step7 Else
Step8 If (the average length increases in this iteration and $Breach_i$ contains no positive numbers)
Step9 Multiply the jth row of *gama* by $e^{-p3 \times Breach_i(j)}$
Step10 Multiply the jth column of *gama* by $e^{-p4 \times Breach_i(j)}$
Step11 End if
Step12 End if
Step13 End for
Step14 End for
Step15 Output resulting matrix *gama*

The negative feedback introduced above can effectively direct and regulate the synergy function among branches, thus controlling tradeoff between length and delay. On the other hand, receiving positive feedback from pheromone, paths with desired objective value and can satisfy the constraints will be repeatedly strengthened and strengthened. At last, under the role of double feedback, an optimal solution with its *Breach* value all negative and closest-to-zero can be found.

8.4 Implementation and Results

Experiments have been conducted to evaluate the performance of our proposed SFB-ACO algorithm, and two groups of experiments are designed and carried out. In the first one, based on the same chip consisting of a certain number of sinks and obstacles, two different scale candidate pools for Steiner points are selected; renewed Prim [48], standard ACO, and SFB-ACO are then applied to optimize the routing using the above two pools, assessments with respect to each are made and roles of synergy function and pool size are carefully discussed. In the second experiment, stringent timing constraints are given according to the Elmore delay tested in the first experiment, and a constraint-oriented feedback is introduced in case of over-constrain, and its effectiveness has been validated through comparisons with AAS.

8.4.1 Parameters Selection

In order to apply relevant algorithms on VLSI routing, several parameters have to be determined. Arguments input related to the chip to be optimized, including source, sinks, obstacles and their positions can be graphically obtained from Fig. 8.8, where the above objects are indicated by red star, red circles, and cyan

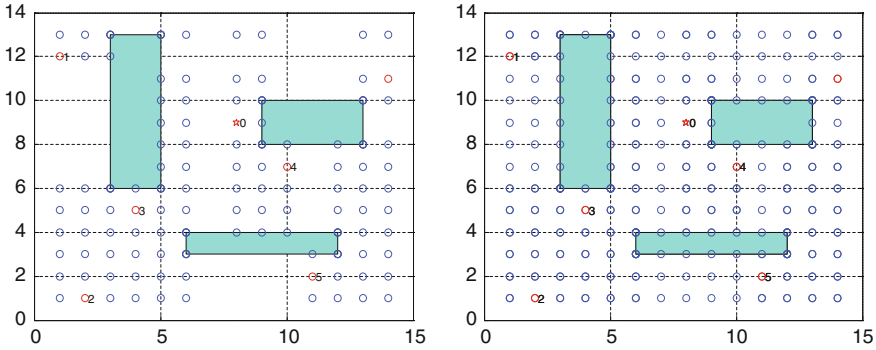


Fig. 8.8 Instance of chip to be optimized and two candidate pools for Steiner points

Table 8.1 Parameters related to algorithms

Parameter	α	β	rho	Q	m	Iter_max	P1	P2	P3	P4
Value	5	20	0.1	2	35	50	0.0000	0.0018	0.0010	0.0000

rectangular. Figure 8.8 also shows two selected candidate pools of different sizes for Steiner points, denoted as pool I and pool II from left to right.

Parameters used in our proposed SFB-ACO are shown in Table 8.1, and those related to the calculation of Elmore delay are summarized in Table 8.2.

8.4.2 Improvement of Synergy

In the first experiment, timing constraints for each sink are set quite loose such that the problem is degraded as a MRSTRO without constraint. Adopting two candidate pools of different sizes for Steiner points, Table 8.3 records the respective results of renewed Prim, standard ACO, and our proposed SFB-ACO, and their optimal routing diagram are given in Figs. 8.9, 8.10, 8.11, 8.12 and 8.13 from top to bottom under two pools. Pool I is a simplified point set of Pool II, in which the points that are not easily accessible, namely, behind obstacles are removed to achieve a lower space complexity. We can see from the data, there is no much difference in the solution quality and convergence rate under two candidate pools. This indeed implies a possibility to reduce algorithm's space complexity while not at the cost of its precision or efficiency. However, this is only valid when leaving the timing constraints aside. If these constraints are stringent, the points behind obstacles may be needed as additional choices for leading a constraint-meet topology of spanning tree. This is the reason that we adopt Pool II in our second experiment.

In Table 8.3, renewed Prim is a kind of greedy algorithm similar to Prim algorithm but with several adjustments mainly considering the Manhattan

Table 8.2 Parameters related to Elmore delay

Parameter	Resistance of				Capacitance of			
	Source (R_{T_0})	Sink (R_{T_i}, $i = 1 : n$)	Node (R_{T_i}, $i > n$)	Edge (R_{rate})	Source (C_{T_0})	Sink (C_{T_i}, $i = 1 : n$)	Node (C_{T_i}, $i > n$)	Edge (C_{rate})
Value	0.4	0.2	0.1	0.1	0.2	0.1	0.05	0.05

Table 8.3 Comparisons of different algorithms under different pools

Algorithm	Minimum length	Average length	Iterations	Elmore delay on each sink					
				Sink 1	Sink 2	Sink 3	Sink 4	Sink 5	Sink 6
Renewed Prim	42	–	–	2.19	1.83	1.65	0.16	2.05	0.32
Adopting pool I for candidate Steiner points (XX elements)									
Standard ACO	43	48	30	1.42	1.44	1.14	0.60	1.44	0.32
SFB-ACO	41	42	12	1.90	1.75	1.53	0.65	1.75	1.03
Adopting pool II for candidate Steiner points (XX elements)									
Standard ACO	42	48	34	1.75	1.66	1.42	0.59	1.45	0.32
SFB-ACO	41	42	18	1.93	1.75	1.51	0.57	1.52	0.32

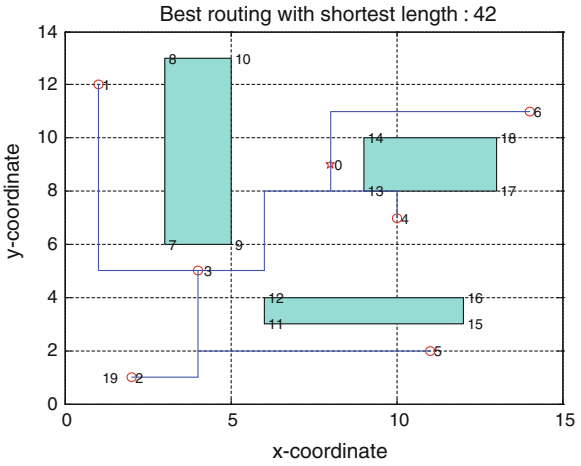


Fig. 8.9 Routing diagram given by renewed Prim

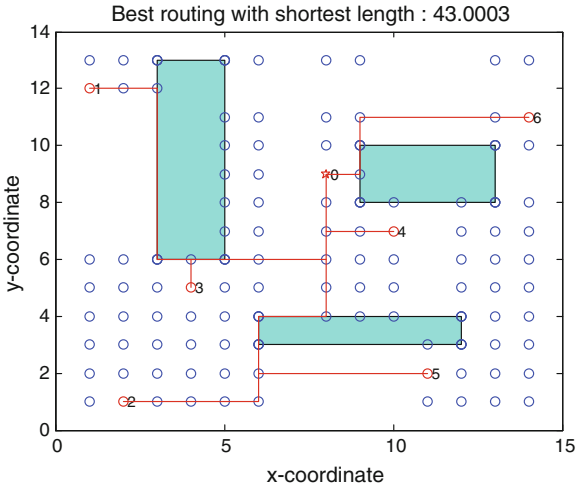


Fig. 8.10 Routing diagram given by standard ACO using pool

architecture and obstacles and its primary mechanism can be described as below. Firstly, starting with a partial tree containing the source, each time we select the sink which has the shortest attainable Manhattan distance to the existing tree. With the selection of sink, the Steiner point can be determined, and therefore an edge between them can be established. The iteration procedure goes on until all sinks have been added to the tree. From above, we know renewed Prim is a relatively deterministic algorithm with quite high efficiency, and that explains why data related to average length and iterations are not recorded in the table. However, in the process of building tree, it is only guided by information given by the added nodes, but without any consideration about the effects it may have on sequential sinks. Its minimum length, 42, though good, is still not the optimal one, compared

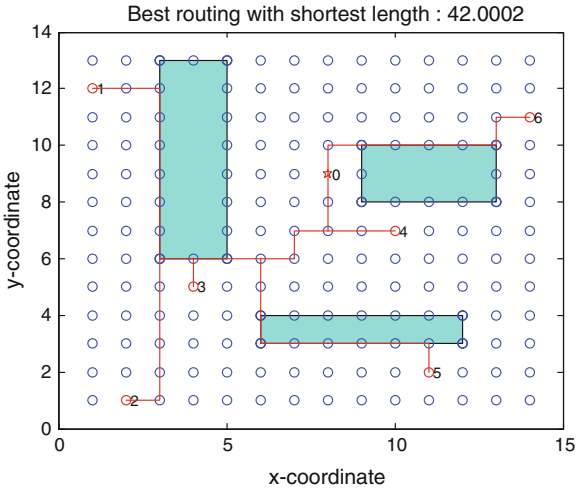


Fig. 8.11 Routing diagram given by standard ACO using pool II

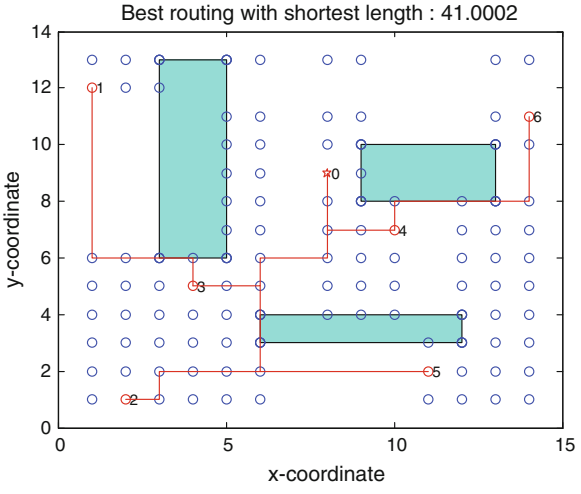


Fig. 8.12 Routing diagram given by SFB-ACO using pool I

to 41 in SFB-ACO. Besides, because of its relative determinacy, it can only obtain solutions with a set of fixed delay on sinks, comparatively, which is absolutely not feasible with stringent timing constraint.

The last two lines in Table 8.3 strongly convince us the advantage of synergy function we introduce in SFB-ACO. Under either pool, SFB-ACO can result in a higher quality of solution and better efficiency of algorithm than standard ACO, 41 versus 43, 41 versus 42, 12 versus 30, and 18 versus 34, respectively. This is because under the function of synergy, branches are no longer independent: they try their best to find ways to join in the tree instead of to reach to the source. Once they merge into another, they just quit travelling and the total length can be

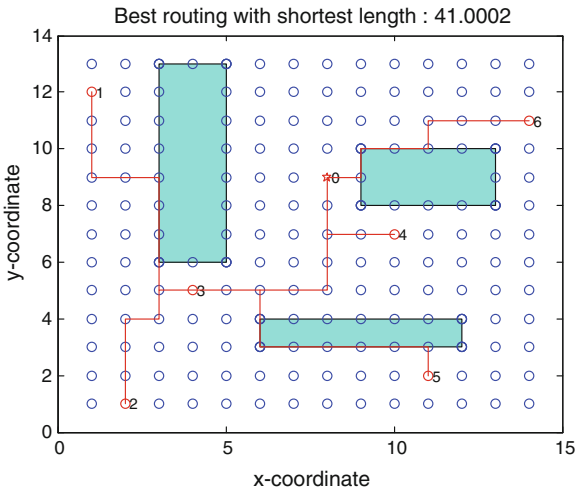


Fig. 8.13 Routing diagram given by SFB-ACO using pool II

reduced. Since the length obtained in their early iterations is already near-to-optimal, the algorithm can converge at a faster rate. Comparing the average length of standard ACO and SFB-ACO, 42 and 48, we also learn that the convergence status in SFB-ACO is better than that in standard ACO. As the iterations goes by, not all solutions can converge into the best one in standard ACO and this is because the pheromone released by the best solution do not have noticeable function on guiding the formatting of its sequential solutions. It, on the other side, implies that merely accounting for pheromone and heuristic information is not enough. Other force, such as our proposed synergy function, is indeed needed.

By comparison of data in Table 8.3 and routing scheme in Figs. 8.9, 8.10, 8.11, 8.12 and 8.13, we also see that same length of two schemes does not necessarily suggest the same topology of spanning tree, not to speak the same delay on each sink. Another purpose for recording Elmore delay in the last couple of columns is for later use as references to giving constraints.

8.4.3 Effectiveness of Constraint-Oriented Feedback

This part will use Pool II for candidate Steiner points, and based on the Elmore delay tested before, a more stringent timing constraint is given. Then through check experiments between conventional AAS and our constraint-oriented feedback, the effectiveness of the proposed practice on preventing over-constrain will be tested.

Table 8.3 illustrates the sink delay of routing solutions with the shortest length under different algorithms. Due to the relatively contradiction between sink delay and tree length, as well as the contrasting relationship between delays on different sinks, we can safely say that delays on some of sinks can be further reduced by increasing the

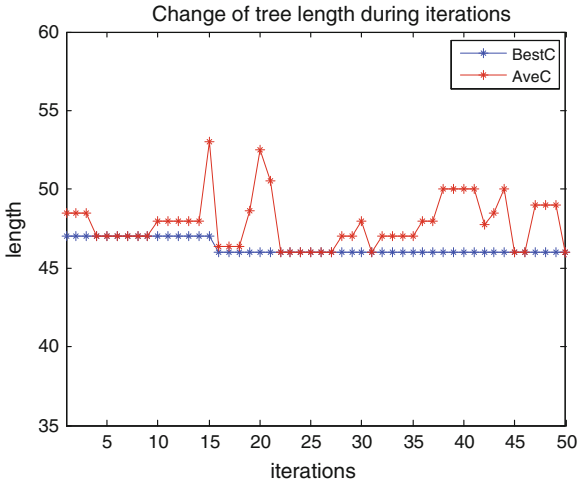


Fig. 8.14 Change of length under AAS

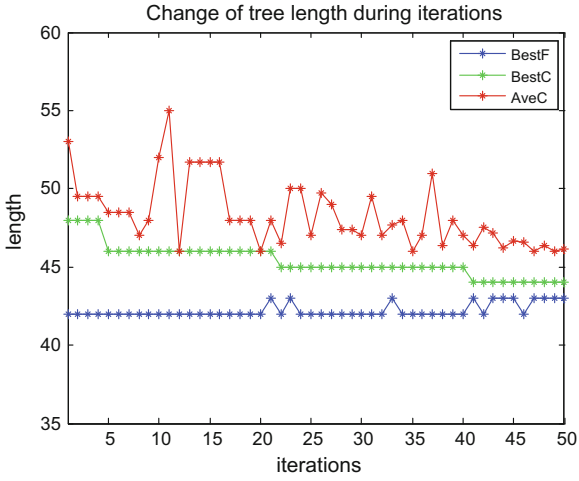


Fig. 8.15 Change of length under constraint-oriented feedback

total length or changing the topology of the final tree. Above analysis leads us to consider setting the timing constraint to be $T_{limit} = [1.5, 1.5, 1.4, 0.4, 1.5, 0.5]$.

Figures 8.14 and 8.15 depict the change of tree lengths during iterations, where *BestF* represents the length of best solutions, regardless of its violation to constraints, *BestC* represents length of best solutions that can meet the constraints, and *AveC* represents average length of solutions that can meet the constraints. If adopting AAS, only solutions under timing constraints will be reserved, and then release pheromone on corresponding paths; this procedure often requires a longer time for curves of *AveC* and *BestC* to meet, and the resulting length is not quite good. Instead, constraint-oriented feedback can take advantage of solutions that have better target value but slightly violate the constraints, by regulating little by little,

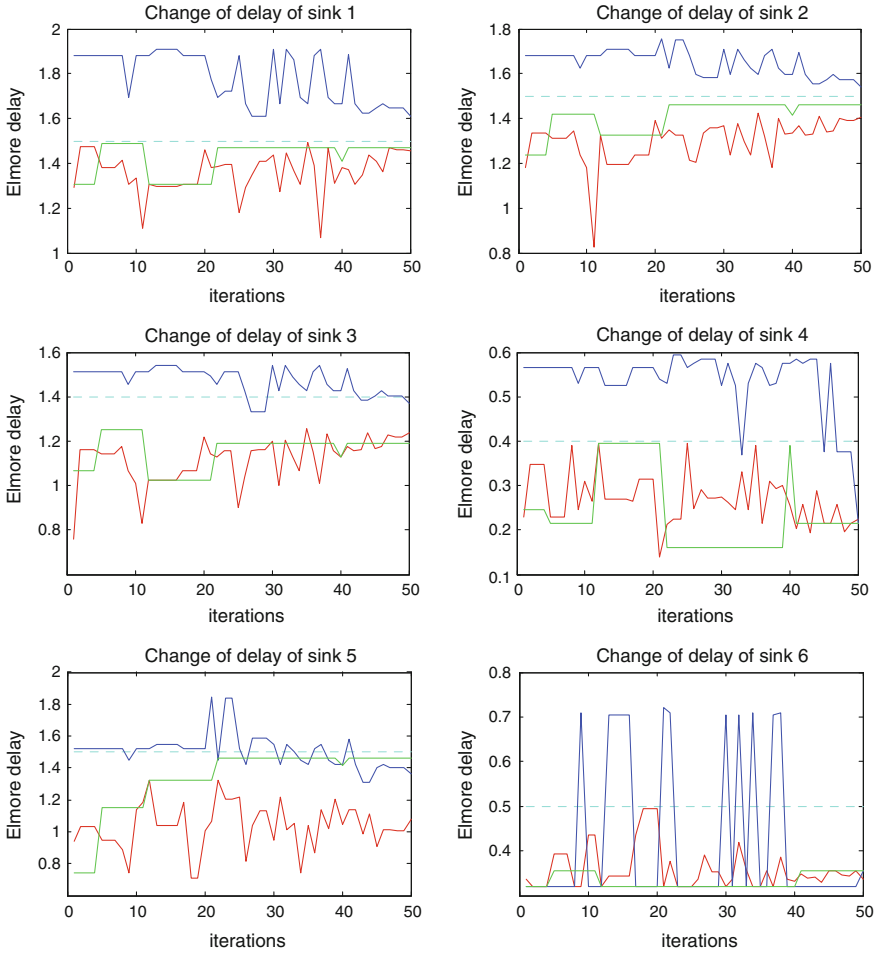


Fig. 8.16 Change of delay on each sinks under constraint-oriented feedback

also requiring quite a long process, can obtain a better solution, 44 compared to 46 in AAS. And finally, three curves in Fig. 8.15 merge together, implying that most of solutions reserved in the last iteration can satisfy the constraints so that the feedback regulation itself is converged. Figure 8.16, which depicts the change of Elmore delay of each sink during iterations, where color cyan, blue, green and, red respectively represent timing constraint, *BestF*, *BestC*, and *AveC*, also explains that points. In the early searching, the blue curves in most figures lie upon the cyan one, indicating that best solution among all feasible ones is somehow against constraints on some of its sinks. In the meanwhile, some of the green curves fall far below the cyan ones, leaving quite an allowance for improving the quality of solutions. As time goes on, the blue curve declines, so is the trend of the red one, while the green curve go through accommodations with others so as to make the overall breach

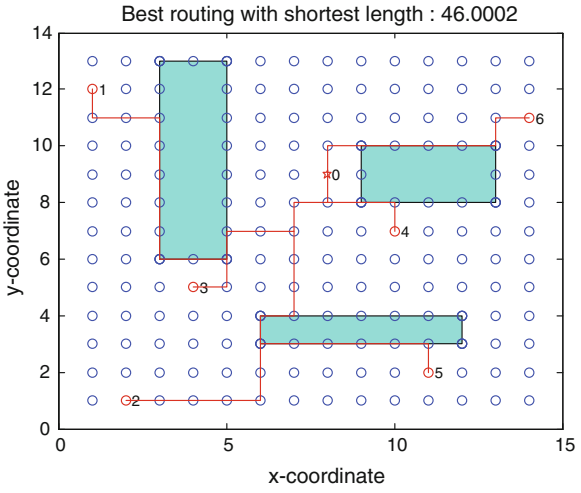


Fig. 8.17 Resulting routing diagram under AAS

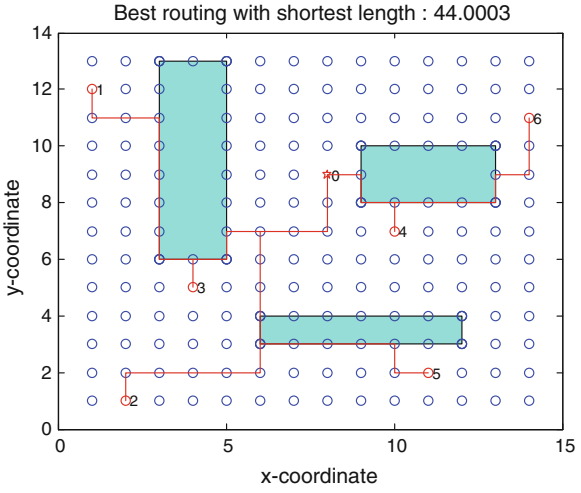


Fig. 8.18 Routing diagram under constraint-oriented feedback

smaller than before. Some sinks, like sink 4, have to larger their breaches, leaving chances for others to minish theirs. This change occurs in iterations around 5, 20, and 40, which corresponds to a step-down in *BestC* curve in Fig. 8.15.

Figures 8.17 and 8.18 give out the final routing diagram under AAS and our constraint-oriented feedback. Table 8.4 records their respective shortest lengths and their corresponding Elmore delays and breaches on each sinks. The one with the shorter length does not necessarily possess the smallest delay on every sinks, but roughly speaking, the breach of it is comparatively closer to zero. Also, shortest length is not automatically equivalent to a near-to-zero value of all elements in its vector *Breach*. Instead, an accommodation between sinks must be considered, and that's the reason why breaches under constraint-oriented feedback

Table 8.4 Comparisons between AAS and constraint-oriented feedback

Method/module	Length	Elmore delay						Breach					
		Sink 1	Sink 2	Sink 3	Sink 4	Sink 5	Sink 6	Sink 1	Sink 2	Sink 3	Sink 4	Sink 5	Sink 6
AAS	46	1.420	1.430	1.100	0.395	1.430	0.320	0.080	0.070	0.300	0.005	0.070	0.180
Feedback	44	1.470	1.460	1.190	0.215	1.460	0.355	0.030	0.040	0.210	0.185	0.040	0.045

are not always smaller than in AAS. Therefore the contradictory relationship discussed before has been once again evidenced, and the effectiveness of our constraint-oriented feedback on preventing over-constrain is convincingly demonstrated.

8.5 Summary

The global routing in VLSI belongs to the multi-terminal path planning, and can be abstracted as a MRSTRO problem. With the coming of submicron age, delay on interconnect can no longer be ignored, which makes the optimization model much different from before. Previous algorithms of constructing Steiner tree are either inapplicable or far from satisfactory. This chapter presented a novel SFB-ACO algorithm, which can serve as a useful tool for net connection with multiple endpoints under constraints. In detail, the main contributions are concluded as follows.

References

1. Chen WK (2004) The electrical engineering handbook. Academic, Burlington, MA
2. Vannelli A (1991) An adaptation of an interior point method for solving the global routing problem. IEEE Trans CAD/ICAS 10(2):193–203
3. Roy JA, Markov IL (2007) High-performance routing at the nanometer scale. In: Proceedings of the International Conference on Computer-Aided Design (ICCAD), San Jose, CA pp 496–502
4. Terlaky T, Vannelli A, Zhang H (2008) On routing in VLSI design and communication networks. Discrete Appl Math 156(11):2178–2194
5. Meindl J (2004) Tyranny of interconnects. In: Proceedings of the International Symposium on Physical Design, pp 18–21
6. Hu J, Sapatnekar S (2002) A survey on multi-net global routing for integrated circuits. Integr VLSI J 31(1):1–49
7. Courant R, Robbins H (1941) What is mathematics? Oxford University Press, New York
8. Hwang FK, Richards DS, Winter P (1992) The Steiner tree problem. Elsevier, Amsterdam
9. Winter P (1985) An algorithm for the Steiner problem in the Euclidean plane. Networks 15:323–345
10. Weng JF, Brazil M, Thomas DA, Zachariasen M (2002) Canonical forms and algorithms for Steiner trees in uniform orientation metrics. Algorithmica, Technical Report: pp 2–22
11. Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations. Plenum, New York
12. Wame DM, Winter P, Zachariasen M (1999) Exact solutions to the large scale plane Steiner tree problems. In: Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms, pp 979–980
13. Snyder TL (1992) On the exact location of Steiner points in general dimension. SIAM J Comput 21(1):163–180
14. Kahng AB, Liu B (2003) Q-tree: a new iterative improvement approach for buffered interconnect optimization. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI, pp 183–188

15. Boese KD, Kahng AB, McCoy BA, Robins G (1995) Near optimal critical sink routing tree constructions. IEEE Trans Comput Aided Des Integr Circ Syst 14(12):1417–1436
16. Hanan M (1966) On Steiner's problem with rectilinear distance. SIAM J Appl Math 14:255–265
17. Kastner R, Bozogzadeh E, Sarrafzadeh M (2000) Predictable routing. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp 110–113
18. Moffett BC (1970) Personal communication. University of Washington
19. Kastner R, Bozorgzadeh E, Sarrafzadeh M (2002) Pattern routing: use and theory for increasing predictability and avoiding coupling. IEEE Trans Comput Aided Des Integr Circ Syst 21(7):777–790
20. Westra J, Bartels C, Groeneveld P (2004) Probabilistic congestion prediction. In: Proceedings of the ACM International Symposium on Physical Design, pp 204–209
21. Hwang FK (1976) On Steiner minimal trees with rectilinear distance. SIAM J Appl Math 30:104–114
22. Nielsen BK, Winter P, Zachariasen M (2002) An exact algorithm for the uniformly-oriented Steiner tree problem. In: Proceedings of the 10th European Symposium on Algorithms. Lecture Notes in Computer Science, vol 2461. Springer, Berlin, pp 760–772
23. Borah M, Owens RM, Irwin MJ (1994) An edge based heuristic for Steiner routing. IEEE Trans Comput Aided Des Integr Circ Syst 13(12):1563–1568
24. Kahng A, Robins G (1992) A new class of iterative Steiner tree heuristics with good performance. IEEE Trans Comput-Aided Des 11:893–902
25. Kahng A, Robins G (1995) On optimal interconnects for VLSI. Kluwer Academic, Boston, MA
26. Griffith J, Robins G, Salowe JS, Zhang T (1994) Closing the gap: near-optimal Steiner trees in polynomial time. IEEE Trans Comput-Aided Des 13(11):1351–1365
27. Chu C (2004) FLUTE: fast lookup table based wire length estimation technique. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design: 696–701
28. C Chu, Y C Wong (2005) Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design. In Proceedings of the 2005 ACM International Symposium on Physical Design, pp 28–35
29. Cho M, Pan DZ (2007) BoxRouter: a new global router based on box expansion and progressive ILP. IEEE Trans Comput-Aided Des Integr Circ Syst 26:2130–2134
30. Pan M, Chu C (2006) FastRoute: a step to integrate global routing into placement. In: Proceedings IEEE/ACM International Conference on Computer-Aided Design, pp 464–471
31. Pan m, Chu C (2007) FastRoute 2.0: a high-quality and efficient global router. In: Proceedings of the 2007 Asia and South Pacific Design Automation Conference, IEEE Computer Society, pp 250–255
32. Lin CW, Chen SY, Li CF, Chang YW, Yang CL (2008) Obstacle-avoiding rectilinear Steiner tree construction based on spanning graphs. IEEE Trans Comput Aided Des Integr Circ Syst 27(4):643–653
33. Feng Z, Hu Y, Jing T, Hong X, Hu X, Yan G (2006) An O(nlogn) algorithm for obstacle-avoiding routing tree construction in the lambda-geometry plane. In: Proceedings of the 46th ACM Annual Design Automation Conference, pp 48–55
34. Shi Y, Mesa P, Yu H, He L (2006) Circuit simulation based obstacle-aware Steiner routing. Proceedings of the 43rd annual conference on Design automation, San Francisco, CA, pp 385–388
35. Shen ZC, Chu CCN, Li YM (2005) Efficient rectilinear Steiner tree construction with rectilinear blockages. In: Proceedings of the 2005 IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp 38–44
36. Hare RM, Julstrom BA (2003) A spanning-tree-based genetic algorithm for some instances of the rectilinear Steiner problem with obstacles. In: Proceedings of the 2003 ACM symposium on Applied Computing, pp 725–729

37. Consoli S, Moreno-Pérez JA, Darby-Dowman K (2010) Discrete particle swarm optimization for the minimum labelling Steiner tree problem. Nat Comput 9(1): 29–46
38. Joobbani R (1985) An artificial intelligence approach to VLSI routing. PhD thesis, Carnegie-Mellon University
39. Lee JW, Choi BS, Lee JJ (2011) Energy-efficient coverage of wireless sensor networks using ant colony optimization with three types of pheromones. IEEE Trans Industr Inform 7(3):419–427
40. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization: artificial ants as computational intelligence technique. IEEE Comput Intell Mag 1(4):28–39
41. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern B Cybern 26(1):29–41
42. Samanta T, Ghosal P, Dasgupta P, Rahaman H (2008) Revisiting fidelity: a case of Elmore-based y-routing tree. In: Proceedings of the 2008 ACM International Workshop on System Level Interconnect Prediction, pp 27–34
43. Dasgupta P (2005) Revisiting VLSI interconnects in deep sub-micron: some open questions. Proceedings of the IEEE 18th International Conference on VLSI Design, pp 81–86
44. McCoy BA, Boese KD, Kahng AB, Robins Gabriel (1995) Near-optimal critical sink routing tree constructions. IEEE Trans Comput-Aided Des Integr Circ Syst 14(12):1417–1436
45. Rubinstein J, Penfield P, Horowitz MA (1983) Signal delay in RC tree networks. IEEE Trans Comput-Aided Des 2:202–211
46. Hou H, Hu J, Sapatnekar SS (1999) Non-Hanan routing. IEEE Trans Comput-Aided Des Integr Circ Syst 18(4):436–444
47. McCoy BA, Boese KD, Kahng AB, Robins G (1993) Fidelity and near-optimality of Elmore-based routing constructions. Proceedings of the IEEE International Conference on Computer Design, pp 81–84
48. Prim RC (1957) Shortest connection networks and some generations. Bell Syst Tech J 36:1389–1401

Chapter 9
A Hybrid RCO for Dual Scheduling of Cloud Service and Computing Resource in Private Cloud

In this chapter, the idea of combining SCOS and OACR into one-time decision in one console is presented, named Dual Scheduling of Cloud Services and Computing Resources (DS-CSCR) [1]. For addressing large-scale DS-CSCR problem, Ranking Chaos Optimization (RCO) is configured. With the consideration of large-scale irregular solution spaces, new adaptive chaos operator is designed to traverse wider spaces within a short time. Besides, dynamic heuristic and ranking selection are hybrid to control the chaos evolution in the proposed algorithm.

9.1 Introduction

Newly developing cloud computing [2, 3] has brought about great benefits to both enterprises and individuals. With advanced technologies of virtualization and service, it incorporates various resources for user on-demand with open interfaces and transparent remote operations. While IBM, Google and Amazon are taking the lead in building general public cloud [4–6] under the modes of SaaS (Software as a Services), IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) [7], many conglomerates have also obtained cost reduction and higher flexibility of resource sharing with the establishment of their own private cloud.

Private cloud of conglomerate usually consists of a set of virtualized distributed infrastructures and application services which are provided by couples of sub-enterprises and partner-enterprises [8, 9], as shown in Fig. 9.1. For outside, such conglomerate could be a large SaaS provider. For inside, it turns to a shared resource pool. In a fairly secure environment, all resources are under the ownership and control of a single administrative domain. On one hand, the virtualization of multiple distributed infrastructures can greatly improve the computing capability for the whole organization with lower-cost. On the other hand, upper layer application services, no matter provided to outside Internet or inside members,

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_9

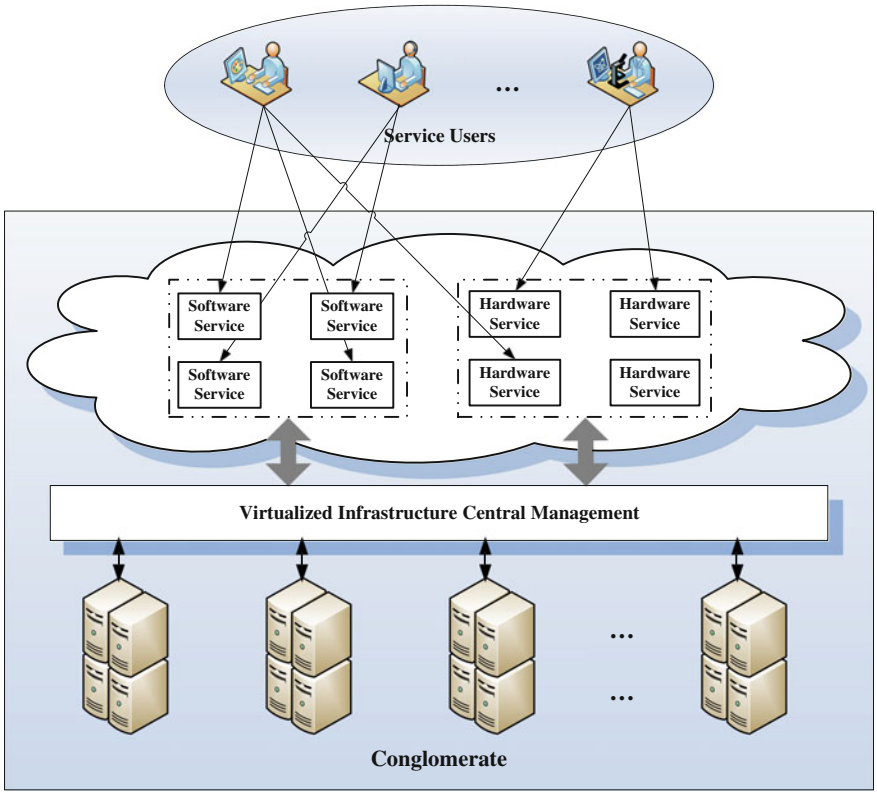


Fig. 9.1 Structure of private cloud and actors in conglomerate

need no longer to be deployed on a fixed computing resource with specific maintenance. Services with central control become more flexible with dynamic allocation. Thus, private cloud in conglomerate also contains two aspects of significance, one is the integration and sharing of underlying distributed infrastructure, another is the flexible deployment and usage of upper layer application services.

Besides, with the development of cloud, the concept of "service" in traditional Service-Oriented Architecture (SOA) is extended from software application to generalized "cloud service" with the inclusion of both software applications and hardware equipments with good interoperability, self-organization and scalability. The properties of cloud services have become more complex and most of them need higher computing ability to drive.

In such environment, when a composite project (which contains a set of tasks) is submitted, the console of conglomerate needs not only to aggregate suitable cloud services with different functionalities and generate service portfolio for user on-demand, but also choose available computing resources to support the running

of cloud services. How to achieve high-quality and low-cost services composition optimal selection (SCOS) and optimal allocation of computing resources (OACR) simultaneously are critical for efficient project execution, green resource sharing and flexible service management.

At present, service composition and computing resource allocation in cloud have been studied preliminarily. Most researches are carried out according to the methodology of cluster computing, grid computing and high performance computing and consider the two problems independently. For one thing, computing availability and communication route of computing resources are analyzed. For another, QoS (Quality of Service) indexes and description languages are also discussed. In general public cloud, SCOS and OACR are performed in two steps and in the charge of different actors. Service providers are not infrastructure providers [3]. However, in private cloud of conglomerates with typical SaaS mode, they would provide suitable service portfolio and deploy corresponding services on their own infrastructure for customers on demand. The actors of SCOS and OACR turn out to be the same one.

With such two-step decision by a single administrator, the properties of upper layer selected cloud services in SCOS will limit the range of the underlying available computing resources for each service in OACR. Better portfolios of cloud services and computing resources are easily overlooked. Furthermore, as all knows, both SCOS and OACR are proved to be NP combinatorial optimization problem. Under the condition of large-scale cloud services and computing resources and complex relationship between them, addressing SCOS and OACR step by step with two different algorithms independently becomes very cumbersome and inefficient.

Therefore, we propose the idea of combining two stages decision-making into one and put forward the concept, Dual Scheduling of Cloud Service and Computing Resource (DS-CSCR), in private cloud of conglomerate. In the guidance of this idea, we analyze the complex features of hardware/software cloud service and computing resource in cloud computing in two levels and explore their mutual relations in-depth. Aiming at green efficient decision, the formulation of DS-CSCR with multi-objectives and multi-constraints is presented in this chapter. Additionally, in order to achieve high efficient one-time decision in DS-CSCR, a new Ranking Chaos Optimization (RCO) is designed in this chapter. Take the advantage of chaotic random ergodicity, this algorithm combines new adaptive chaos optimal strategy with ranking selection and dynamic heuristic mechanism to balance the exploration and exploitation in optimization. With adaptive control of chaotic sequence length, it's especially good at searching in large-scaled irregular solution space and shows remarkable performance for addressing DS-CSCR compared with other general intelligent algorithms.

9.2 Related Works

Nowadays, the most commonly used and analyzed cloud computing platforms are "Google cloud computing" platform, Amazon "elastic cloud" platform and IBM "blue cloud" platform. Private cloud with closed sharing are researched less and attracted criticism owing to the less hands-on management [4]. But it can notably reduce the cost of resources and improve the quality of services in large conglomerate. After years of development, large enterprises, academic institutions and new emerging internet service providers are building their own cloud platform, too, such as Eucalyptus, Red Hat's cloud, OpenNebula and so on. Though various platforms differ on their usage mode and openness, most of them share the same key technologies and target of resource sharing.

In cloud computing, two crucial optimization factors in determining resource sharing efficiency and platform application performance are SCOS and OACR exactly.

In recent years, researches on service composition are generally based on the environment of grid computing and other SOA mode [10]. These researches spread from service description language, service QoS indexes [11], reliability and trust evaluation [12], and optimal selection of services [13] and so on. Since cloud computing mode has been proposed, the concept and content of cloud service are broadened. According to the characteristics of cloud computing, semantic properties of cloud service are studied [14]. The classification, management, provision, storage and evaluation of cloud services are investigated widely. Pre-decision and online-decision of SCOS are also deliberated in different ways, such as [15]. Among these, QoS indexes of cloud services are discussed most widely. From the perspective of non-functional properties of cloud services, the existing indexes consider no more than cost, time and reliability factors. It's hard to describe various cloud services with different classification and attributes in a unified form. Thus the existing QoS indexes can't satisfy all types of cloud services.

For computing resource allocation, traditional researches mostly focus on the modeling and evaluation of computing resources based on homogeneous/heterogeneous cluster systems or distributed grid computing systems [16]. User's demand for resources, resources' costs and computation and communication capabilities of resources are the major considerations among these studies. In cloud computing mode, virtualization is the main support of flexible resource sharing [17]. In this context, Endo et al. introduced the concept, classification of resource allocation in distributed cloud [18]. Ma et al. [19] and Xiong et al. [20] investigated the management of cloud computing resources based on ontology and virtualization respectively. Zhang et al. [21] proposed a method for the deployment of upper layer software cloud services from virtual machines. Ghanbari et al. [22] have studied the feedback-based optimization problem including the allocation of resources especially in private cloud. Besides, considering the virtual division of computing resources and its influences on the quality of cloud services, researchers also built new models for computing resources from the rules,

reliability and dynamic partition point of view, and so on, and presented various methods to solve OACR problem in cloud computing [23, 24]. Most of these studies concentrated on the expansion of characteristics of computing resources based on traditional models and the algorithm designing for OACR in cloud computing. However, the mutual relations between cloud services and the underlying computing resources and the influence of virtualization on quality of cloud services, as two of the key factors in cloud computing, have not been studied.

In addition, SCOS and OACR are both combinatorial optimization problems. For this kind of problems, the most widely used algorithms are intelligent algorithms due to its NP complexity. It includes Genetic Algorithm (GA) [25], Particle Swarm Optimization (PSO) [26] and so on and has the virtues of brachylogy, universality and rapidity. According to different specific problems, abundant researches mainly focus on the balance of exploration and exploitation in searching process based on evolutionary iteration of population and presented many kinds of improved hybrid intelligent algorithms such as [27]. Nevertheless, these improved hybrid intelligent algorithms are mostly problem-dependent with local convergence more or less. For addressing large-scaled DS-CSCR problem in private cloud of large conglomerate with irregular solution space efficiently, the design of high performance intelligent algorithm is imperative.

9.3 Motivation Example

Currently, the concept of cloud is studied and applied in almost every field. Based on the technology of cloud computing, manufacturing equipments and simulation software as cloud services can be realized [28, 29]. Various software and hardware can by dynamically shared for product customization of both inside or outside organizations without repeat-purchase. Under this background, we use “the design and NC (Numerical Control) machining of a complex surface part in conglomerate cloud” as a case to describe the whole process from tasks’ submission to tasks’ execution. As shown in Fig. 9.2, it can be divided into five sub-tasks: (1) technical and mathematical analysis, (2) CAD modeling and NC programming, (3) verification simulation and post-processing, (4) first NC machining and measuring, and (5) batch production.

During this process, task (1), (2) and (3) can be implemented directly by manufacturing software cloud services, such as CATIA, MasterCAM or Pro/E, etc., and task 4 and task 5 can be executed by manufacturing hardware cloud service with users’ supervision and control, such as 3-axis, 4-axis or 5-axis linkage CNC (Computer Numerical Control) machines, etc. When user submitted the tasks of designing and machining a customized part, four steps are needed to be done by centre console: (1) Requirement analysis of tasks, (2) Services composition optimal selection, (3) Optimal allocation of computing resources, (4) Execution.

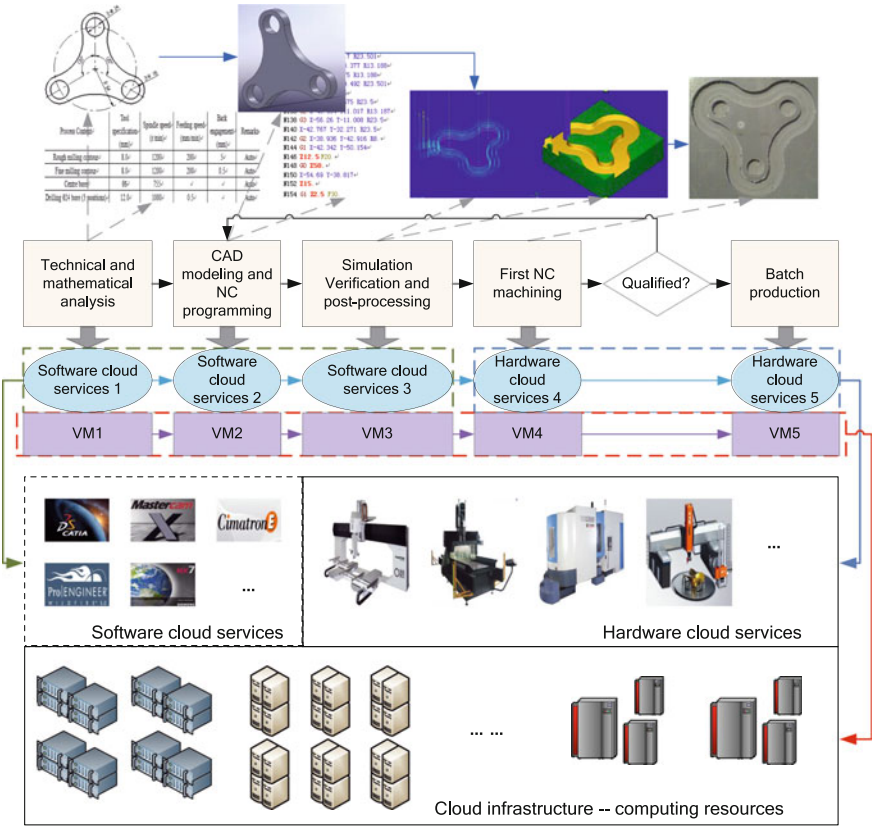


Fig. 9.2 The design and NC machining of a customized part in conglomerate cloud

The scheduling of computing resources totally depends on the corresponding upper layer selected cloud services. With the distributed characteristics of services and infrastructures, the available computing resources are reduced and the OACR are constrained by the upper layer decision. For example, for task (4), assume the suitable CNC hardware service No. 1 and No. 2 are provided in Location A and Location B respectively. CNC service No. 1 is with higher QoS than CNC service No. 2. But the idle computing resources in Location A are less than Location B. If CNC service No. 1 is selected for task 4 in step 3, the low computing ability of computing resource in Location A and the remote communication overhead of computing resource in Location B would both cause the low execution efficiency of CNC service No. 1. If we select CNC service No. 2, the better available adjacent computing resource would then improve the overall execution efficiency of task 4. However, the decision of SCOS in step 2 usually disregards the influence of the underlying support computing resources due to the traditional binding mode of service and infrastructure. The latter strategy of choosing MasterCAM service

No. 2 is then overlooked. At this time, you might say, if SCOS and OACR are performed at the same time, then bad decision won't be happened.

Therefore, in order to reduce the time and improve the quality of decision, we merge SCOS and OACR into one dual-scheduling decision. With the purpose of efficient DS-CSCR decision, the following three issues are needed to be studied.

(1) QoS indexes of software/hardware cloud services and computing resources respectively and the mutual relation between them;
(2) The problem formulation of DS-CSCR with multi-objectives and multi-constraints in private cloud;
(3) The efficient scheduling algorithm for addressing large-scale DS-CSCR problem.

This chapter will directly focus on these three issues.

9.4 Problem Description

9.4.1 The Modeling of DS-CSCR in Private Cloud

In conglomerate, services and the support infrastructures are provided by distributed sub-enterprises and controlled by central head. Traditionally, service provider usually deploy the service to a fixed computer, put service and computing resource together to ensure the quality of service. The support computing resources are always occupied by fixed service and needed specific maintenance. With new cloud mode, services can be encapsulated and registered to cloud and deployed to virtual machines dynamically. Through the collaborative development of upper layer applications and underlying resources, all of the resources can be shared flexibly on-demand with more energy-saving, higher redundancy and reliability.

Moreover, based on such a flexible environment, cloud services with the support of VMs contain not only software cloud services, but also hardware services with further expansion. For hardware cloud services, the computing resources are no longer support carriers, but controlling and monitoring facilities for these manufacturing equipments.

(1) The characteristics and QoS indexes of cloud services
From the perspective of QoS evaluation, only simplified quantitative cost, time and reliability cannot comprehensively summarize the characteristics of software/hardware cloud services and their requirement for VMs' performance. With the consideration of the difference between software and hardware cloud services and their demands for VM configuration, this section gives new evaluation indexes for software/hardware cloud service and virtual resources respectively.

(a) The characteristics and QoS indexes of software cloud services

Software applications in cloud computing are running with the support of VMs. Each software service is deployed to a single VM and mapped to a corresponding computing resource. Thus the minimum requirements of VM which represents the required volume of services should be defined to facilitate the allocation of computing resources. Based on the functional description of services, we consider mainly the following non-functional factors of software cloud services in this chapter.

- ***s***—service execution efficiency under the minimum required configuration of VM;
- ***c***—the rent cost of service;
- ***r***—trustiness of service, which is the ratio of the success execution time and the total execution time;
- ***v***—the minimum required speed of VM.

Remarks The performance of the required VM is determined by many factors, such as the CPU and memory of the corresponding computing resources. In a computer, the speed of CPU is in proportion to the power supply voltage [30]. It's a constant value. The speed of VM can mainly be calculated by the number and speed of occupied CPUs. So that the minimum required speed of VM is adopted here for evaluation. The higher the speed of VM is, the faster the service runs.

(b) The characteristics and QoS indexes of hardware cloud services

Unlike the software service, hardware service is energy-consuming and needs supervision or control during execution. Real-time supervision or control will produce large amount of communication and increase service execution time (i.e. the time-consumption of data transmission). Different hardware service needs different amount of supervision and control. For this reason, based on the above four factors of software service, two more factors need to be considered.

- ***s***—service execution efficiency under the minimum required configuration of VM;
- ***c***—the rent cost of service;
- ***r***—trustiness of service, which is the ratio of the success execution time and the total execution time;
- ***v***—the minimum required speed of VM;
- ***e***—the average energy-consumption of hardware service;
- ζ—the average control rate, which is the ratio of the amount of control commands and the amount of tasks;
- η—the transmission rate between VM (computing resources) and hardware service.

Remarks For hardware services, there are two conditions of control. One is inputting all control commands beforehand, and then executing tasks without interaction. Another is controlling during execution. Owning to the large amount of task in hardware service, ζ in the first condition can usually be ignored

(i.e. $\zeta = 0$). We mainly focus on the second condition. Besides, if the hardware service needs no control or supervision any more, then $\zeta = 0$, too.

Usually, the transmission path of the control commands of software service is "user—VM", while which of hardware service is "user—VM—hardware service". Without the consideration of task interactions and energy-consumption of VMs, if the amount of submitted task is W, the total execution time T, the total cost C and the total energy-consumption E of the software and hardware service can be calculated as follows respectively.

For software services,

$$T = \frac{W}{s} \tag{9.1}$$

$$C = Tc = \frac{cW}{s} \tag{9.2}$$

For hardware services,

$$T = \frac{W}{s} + \frac{W\zeta}{\eta} = W\frac{\eta + s\zeta}{s\eta} \tag{9.3}$$

$$C = Tc = cW\frac{\eta + s\zeta}{s\eta} \tag{9.4}$$

$$E = Te = eW\frac{\eta + s\zeta}{s\eta} \tag{9.5}$$

(2) The characteristics and QoS indexes of VMs

VMs are the virtual division of the underlying computing resources. The performance of VM are mainly embodied in the running speed, transmission rate and energy consumption of the corresponding computing resources. It's still hard to locate one VM into multiple computers by existing technologies of virtualization. Hence, we assume each VM maps into only one physical node. In accordance with the characteristics of cloud services, we primarily concentrate on four factors below.

- ***p***—the running speed of VM, which depends on the occupancy rate and the speed of CPUs;
- ***q***—the transmission rate of VM;
- ***g***—the average energy-consumption of VM;
- ***f***—the failure probability of VM;
- ***u***—the recovery time of VM when fails.

Remarks q reflects the transmission rate between the occupied physical computing resources and the objects. If the transport object and the VM are in the same local network, then evaluate the transmission rate by local bandwidth. Else, the transmission rate is evaluated with the synthetic consideration of the transport

object, the central console and the VM itself. Besides, the energy function of CPU per unit time can be represented as [29]: $P_0 = AV^2f + Z$. Where A and Z are constant, V is the power supply voltage and f is the dominant frequency. Thus g is in proportion to p, too. In cloud platform, the way to handle the failures of physical nodes is usually dynamic migration of VMs. So, u is no longer the recovery time of the corresponding computing resource but the dynamic migration time. Computing resources with low reliability can easily cause dramatically increase of task execution time, cost and energy consumption.

Let the task execution time in the corresponding VM without failure be t, the average task execution time of VM can be evaluated as:

$$\tilde{t} = t(1-f) + (t+u)f = t + fu \tag{9.6}$$

Assume the set of the predecessor nodes of the task i to be $\mathbf{L_i}$, and the input communication amount from the predecessor node j is U_{ij}, then the total communication time between the task and its predecessor nodes are:

$$U = \max_{j \in \mathbf{L_i}} \frac{U_{ij}}{q_j} \tag{9.7}$$

If the performance of VM can satisfy the minimum requirement of service, then the total execution time T, the total cost C and the total energy consumption E of the task can be calculated as follows.

(a) If the selected service is software cloud service, then

$$T = \frac{vW}{ps} + U + fu \tag{9.8}$$

$$C = Tc = \left(\frac{vW}{ps} + U + fu\right)c \tag{9.9}$$

$$E = Te = \left(\frac{vW}{ps} + U + fu\right)e \tag{9.10}$$

(b) If the selected service is hardware cloud service, then

$$T = \frac{vW(\eta + s\zeta)}{ps\zeta} + U + fu \tag{9.11}$$

$$C = Tc = \left(\frac{vW(\eta + s\zeta)}{ps\zeta} + U + fu\right)c \tag{9.12}$$

$$E = T(g+e) = \left(\frac{vW(\eta + s\zeta)}{ps\zeta} + U + fu\right)(g+e) \tag{9.13}$$

9.4.2 Problem Formulation of DS-CSCR in Private Cloud

According to the analysis of the characteristics and QoS indexes of cloud services and virtual resources, the abstract formal description of cloud services, VMs and computing resources are elaborated in this Section.

Definition 1 The set of tasks in cloud computing environment can be presented as a directed acyclic gragh (DAG) $G = (N, W, U, H_t, H_c, H_e, H_r)$. Where

- The set $\mathbf{N} = \{N_i | i = 1 : n\}$ represents tasks with serial numbers, where n is the total number of tasks.
- The set $\mathbf{W} = \{W_i | i = 1 : n\}$ indicates the size of tasks.
- The set $\mathbf{U} = \{U_{ij} | i = 1 : n, j = 1 : n\}$ represents the communication relationships among tasks, where U_{ij} reflects the communication from task N_i to task N_j. We should note that $U_{ij} \neq U_{ji}$. If there's no communication between the two tasks, then $U_{ij} = 0$.
- $\mathbf{H_t} = \{H_t(i) | i = 1 : n\}$, $\mathbf{H_c} = \{H_c(i) | i = 1 : n\}$, $\mathbf{H_e} = \{H_e(i) | i = 1 : n\}$ and $\mathbf{H_r} = \{H_r(i) | i = 1 : n\}$ represent the lowest time, cost, energy and reliability requirements of tasks respectively.

Besides, let the predecessor tasks set of N_i be $\mathbf{L_i}$, and the successor tasks set be $\mathbf{R_i}$. The node with no predecessor task $\mathbf{L_i} = \emptyset$ is named *source* node, and the node with no successor task $\mathbf{R_i} = \emptyset$ is called sink node. All tasks strictly observe the tasks' priority rules, that is to say, a node can only be started after all output communication data of its predecessor tasks are obtained.

According to the QoS indexes analyzed in the previous sections, the general model of cloud computing can be defined as follow.

Definition 2 The software/hardware cloud services in cloud computing mode can be presented respectively as

$$S : \begin{cases} \text{software service} : \ S_1 = (s, c, r, v) \\ \text{hardware service} : \ S_2 = (s, c, r, v, e, \zeta) \end{cases}$$

$\mathbf{S_1} = \{s_1(i) | i = 1 : n_{s_1}\}$ represents the set of software cloud services, where the number of services is $n_{s_1} = |\mathbf{S_1}|$. $\mathbf{S}_2 = \{s_2(i) | i = 1 : n_{s_2}\}$ represents the set of hardware cloud services, where the number of services is $n_{s_2} = |\mathbf{S}_2|$. Therefore the total number of cloud services is $n_s = n_{s_1} + n_{s_2}$. In the definition, s, c, r, v, e, and ζ represents the execution efficiency, rent cost, reliability, the minimum required speed of VM, energy-consumption and the average control rate of cloud services respectively. All of these attributes stored according to the type and the serial number of services.

Because the performance of VM is decided by the corresponding computing resources, so this chapter just define the formal description of computing resources as follow.

Definition 3 The computing resources in cloud computing mode can be presented as $P = (x, \varphi, \phi, \sigma, f, \lambda)$, where

- $\mathbf{P} = \{P_{kl} | k = 1, 2, \ldots, d, l = 1, 2, \ldots, m_k\}$ indicates the computing resources with different groups and different serial number, where k is the group number of the whole set, l is the number of computing resources in each group and d is the number of groups.
- $\mathbf{x} = \{x_{kl} | k = 1, 2, \ldots, d, l = 1, 2, \ldots, m_k\}$ represents the speed of computing resources. It's related to the configuration characteristics of these computers.
- $\Psi = \{\varphi_{kl} | k = 1, 2, \ldots, d, l = 1, 2, \ldots, m_k\}$ means the bandwidths of computing resources in local networks, and $\Phi = \{\phi_k | k = 1, 2, \ldots, d\}$ be the bandwidths between the switches of various sub-infrastructure groups and cloud centre console.
- $\boldsymbol{\sigma} = \{\sigma_{kl} | k = 1, 2, \ldots, d, l = 1, 2, \ldots, m_k\}$ represents the average energy-consumption per unit time of these computing resources. According to the analysis above, σ_{kl} is in proportional to x_{kl}.
- $\mathbf{f} = \{f_{kl} | k = 1, 2, \ldots, d, l = 1, 2, \ldots, m_k\}$ means the failure probability of each computing resource. This factor is changed after each time of task execution.
- $\boldsymbol{\lambda} = \{\lambda_{kl} | k = 1, 2, \ldots, d, l = 1, 2, \ldots, m_k\}$ represents the number of task loads in each computing resource at present. It changed during task execution. If multiple VMs map into one single computing resource, the running speed of the resource will be dramatically declined. For simplified the evaluation, we assume the VMs share the same computing resource with average division.

In the definition of computing resources, the failure recovery time is not defined. Because of the dynamic migration in cloud computing system, we assume the average dynamic migration time (i.e. the recovery time) as a constant $u = Const$.

For two tasks N_i and N_j, if the support VM are v_i and v_j, and the allocated computing resources are P_{kl} and $P_{k'l'}$, the running speed of v_i and v_j can be expressed as $p_i = x_{kl}/\lambda_{kl}$ and $p_j = x_{k'l'}/\lambda_{k'l'}$. If the selected computing resources are in the same group, i.e. $k = k'$, the transmission rate is $q_{ij} = \min(\varphi_{kl}, \varphi_{k'l'})$. If the allocated computing resources are distributed, the transmission rate can be represented as $q_{ij} = \min(\phi_k, \phi_{k'})$. In addition, the energy-consumption of the two VMs are $g_i = \sigma_{kl}/\lambda_{kl}$ and $g_j = \sigma_{k'l'}/\lambda_{k'l'}$. And the rent cost, failure probability and recovery time of VMs are defined the same as the attributes of computing resources.

Corresponding to Fig. 9.3, the DS-CSCR model can be defined as a quadric-tuple $M = (G, S, V, P)$. Based on the above definitions, the decision of DS-CSCR can be made and evaluated with multi objectives of the lowest execution time, energy-consumption and cost and the highest reliability for tasks.

Take the serial tasks as a case, let the number of tasks be n, the type of the selected cloud service for each task N_i is y_i. y_i can be 1 or 2 and represents

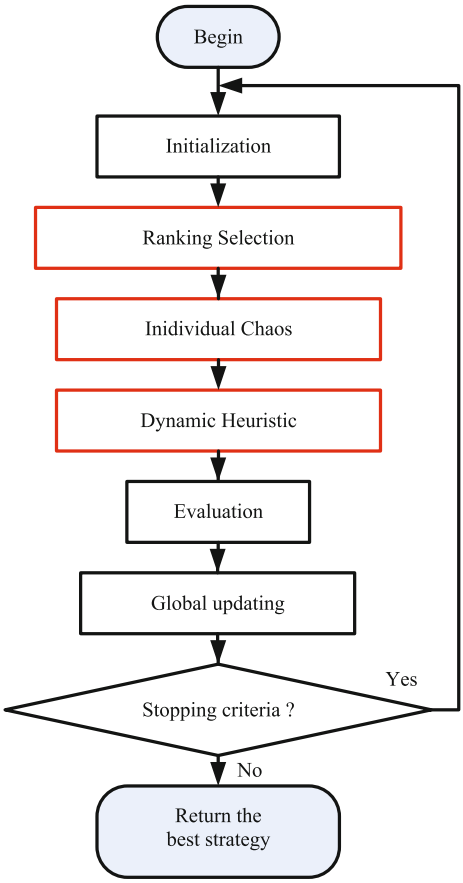


Fig. 9.3 The flowchart of RCO

software and hardware cloud service respectively. So that the serial number of the selected service is $S_{y_i}(i)$. Assume the allocated computing resource for the support VM v_i of each task is $P_{k_i l_i}$. Then the overall optimal objectives and constraints can be calculated as follows.

$$MAX\ Objective\ Function = w_1 \prod_{i=1}^{n} R_i + w_2 / \sum_{i=1}^{n} T_i + w_3 / \sum_{i=1}^{n} C_i + w_4 / \sum_{i=1}^{n} E_i \tag{9.14}$$

The variables in the objective function are calculated according to Table 9.1.

Table 9.1 The calculation of elements in the objective function

Variables	Software services	Hardware services
R_i	$r_{s_1(i)}$	$r_{s_2(i)}$
T_i	$W_i \frac{v_{s_1(i)}\sigma_{k_i l_i}}{x_{k_i l_i} s_{s_1(i)}} + \max_{j \in pred(i)} \frac{U_{ij}}{q_{ij}} + uf_{k_i l_i}$	$W_i \frac{v_{s_2(i)}\lambda_{k_i l_i}(\phi_{k_i} + s_{s_2(i)}\zeta_{s_2(i)})}{x_{k_i l_i} s_{s_2(i)}\phi_{k_i}} + \max_{j \in pred(i)} \frac{U_{ij}}{q_{ij}} + uf_{k_i l_i}$
C_i	$T_i c_{s_1(i)}$	$T_i c_{s_2(i)}$
E_i	$T_i \frac{\sigma_{k_i l_i}}{\lambda_{k_i l_i}}$	$T_i \left(\frac{\sigma_{k_i l_i}}{\lambda_{k_i l_i}} + e_{s_2(i)} \right)$

The main constraints of DS-CSCR are shown as following

$$\forall i \in [1, n] \qquad 0 < \rho_i < 1 \tag{9.15}$$

$$\forall k \in [1, g], l \in [1, m_k] \qquad \sigma_{kl} \geq 0 \tag{9.16}$$

$$\forall i \in [1, n] \qquad T_i < H_t(i),\ C_i < H_c(i),\ E_i < H_e(i),\ R_i < H_r(i) \tag{9.17}$$

The first constraint means that the occupancy rates of VMs in computing resources are no less than 0 and no more than 1, that is to say, one VM can only be allocated in one computing resource with full occupancy at most. The second constraint indicates that the load of computing resources must be no less than 0. When $\sigma_{kl} = 0$, the computing resource is idle. When $0 < \sigma_{kl} < 1$, the computing resource is not fully occupied, the running speed can be hold. However, when $\sigma_{kl} \geq 1$, the tasks need to be executed in queue, the running speed of computing resource will be dramatically decreased. The third constraint represents that each attributes of cloud services and computing resources must satisfy the lowest requirement of tasks.

9.5 Ranking Chaos Algorithm (RCO) for DS-CSCR in Private Cloud

From the above analysis it's clear that the model of DS-CSCR is more complex than the traditional SCOS and OACR. The upper layer cloud services and the underlying computing resources interact with each other. Their complex attributes together directly determine the efficiency of task execution. In large-scale solution space, it's hard to find optimal solution of DS-CSCR by a deterministic algorithm. The general methods for solving these kinds of problems are searching for sub-optimal solutions by intelligent algorithms, such as GA, PSO and ACO and so on. ACO is designed particularly for path optimization. PSO is presented for continuous numerical optimization. GA is more universal but with serious local convergence. In the condition of complex mutual relations among the attributes of the problems with large-scaled irregular solution space, these typical algorithms are quite unsuitable.

Therefore, a new RCO is presented in this chapter for DS-CSCR. The flowchart of this algorithm is shown in Fig. 9.3. It contains three main operators: ranking selection operator, adaptive chaos operator and dynamic heuristic operator. All of them can be executed independently and hybrid arbitrarily. Their initialization (coding scheme), operators and evolutionary strategy for solving DS-CSCR are elaborated as follows.

9.5.1 Initialization

Usually, initialization in intelligent algorithm is very important. It determines the initial location and the coding scheme of population. The initial location ways of population include regular generation and random generation, and so on. For DS-CSCR, the solution space is quite complex, so that the random initialization scheme is selected in this chapter.

Additionally, different coding style has different contribution to algorithm. Coding scheme in intelligent algorithm not only directly reflects the characteristics of the problems, but also affects the performance of the operators. Suitable coding scheme can even improve the searching capability of algorithms. In this chapter, the real number coding scheme is adopted because of its characteristics of simplicity and intuitive.

Specifically, for the above mentioned DS-CSCR model, both service-genebit which represents the selected cloud services and resource-genebit which represents the allocated computing resources are needed to be set. One task corresponds to two genebits. Thus the real number coding is the most intuitive and space-saving scheme for DS-CSCR. When a set of tasks are submitted to cloud system, the system should choose suitable cloud services and computing resources with specific serial numbers at the same time. Assume the length of gene code be twice of the number of tasks, as shown in Fig. 9.4. Each two genebits represent the serial number of the selected service and the allocated computing resource for the corresponding task. It briefly demonstrates the relationship between cloud service and computing resource and makes the optimal process more convenient.

9.5.2 Ranking Selection Operator

In most chaos-based optimizations, chaotic operator is based on the individuals regardless of whether they are good or bad. In this case, the algorithm is easy to trap into bad conditions with large randomly searching range and extremely strong diversity. To obtain better seeds for chaotic random ergodicity, selection before it is needed.

The most commonly used selection operator in GA is roulette wheel selection. With high randomness, bad individuals may be selected more than good ones,

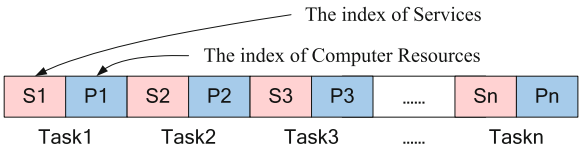


Fig. 9.4 The real number coding scheme for DS-CSCR problem

higher diversity can be achieved in population. But high diversity has been implemented by chaos and what we need before chaos is just a set of good seeds. In this condition, roulette wheel selection becomes unsuitable. To make sure the high quality of good individuals (i.e. seeds), a dynamic ranking selection operator is designed in this section.

Normally, ranking selection means selection according to the descending sort of individual fitness values under a constant proportion. That is to say, the numbers of individuals from best to worst are in arithmetic sequence. Here we adopt quick sort algorithm with the computation complexity O($n$log$n$). Let $\mathbf{I} = \{I_i | i = 1, 2, N\}$ be the population with N individuals, and I_i in the population be the ith individual. Assume the sorted population to be $\mathbf{I}' = \{I'_i | i = 1, 2, N\}$ with the fitness value $F'_N < F'_{N-1} < \cdots < F'_1$. Define $P_{selection}$ to be the percentage of individuals to be selected on the whole. If $P_{selection} = 1$, then all individuals are selected at least once, if $P_{selection} = 0.5$, then only the first half individuals are selected, the other half individuals would not be selected any more. It represents the selection range in the sorted population. Thus the worst individual to be selected is the Kth individual where $K = NP_{selection}$. Under the selection range, let the number of times that the best individual to be selected as θ_1 and the number of times that the worst individual to be selected as θ_K. Then the difference between the numbers of two adjacent individuals can be calculated as follow.

$$\Delta\theta = \theta_{i-1} - \theta_i = \frac{\theta_1 - \theta_K}{K - 1} \tag{9.18}$$

$$\theta_i = \theta_1 - \Delta\theta(i-1) = \theta_1 - (i-1)\frac{\theta_K - \theta_1}{K-1} \quad \text{where} \quad 1 \le i \le K \tag{9.19}$$

It can be seen that $\sum_{i=1}^{K} \theta_i = N$. Therefore, we can deduce that,

$$\theta_1 + \theta_K = \frac{2N}{K} \tag{9.20}$$

Let $\theta_K = 1$, then

$$1 = \theta_K \le \theta_1 \le \frac{2N}{K} - 1 \tag{9.21}$$

To make the selection adaptively, a function for calculating θ_1 in the ranking selection is defined as follow.

$$\theta_1 = 1 + \left(\frac{2N}{K} - 2\right)\frac{F_{average}}{F_{best}} = 1 + \left(\frac{2N}{K} - 2\right)\frac{F_{average}}{F_1'} \tag{9.22}$$

$$\Delta\theta = \left(\frac{2N}{K} - 2\right)\frac{F_{average}}{F_1'}\frac{1}{(K-1)} = \frac{2(N-2K)F_{average}}{K(K-1)F_1'} \tag{9.23}$$

where $F_{average}$ represents of the average fitness value of the whole population. Thus the much closer $F_{average}$ and F' are, the bigger θ_1 is, the bigger the number of times the better individuals to be selected. Otherwise, the number of times the worse individuals would be bigger and the selection of K individuals becomes more balance. The pseudo-code of this operator is shown below as Algorithm 1.

Algorithm 1: Ranking Selection Operator
Ranking_Selection (I)
Define the selection range according to $P_{selection}$
Sort *I* with quick sort algorithm and stored as $\mathbf{I}'$
Calculate the number of times of I_1 to be selected, $\theta_1 = 1 + (2N/K - 2)F_{average}/F_1'$
Calculate $\Delta\theta = 2(N-2K)F_{average}/K(K-1)F_1'$
Calculate $\theta_2, \theta_3, \cdots, \theta_K$ for other $K-1$ individuals
Select *N* individuals according to $\theta_1, \theta_2, \cdots, \theta_K$ and generate new *I*

9.5.3 Individual Chaos Operator

Chaos is a universal non-linear phenomenon. It has the characteristics of strong randomness and internal regularity. With the generation of logistic chaos sequences, it can traverse almost all states in a certain range without duplication and cause great changes in output with rich dynamism. Thus it can improve population diversity in many typical intelligent algorithms and help them to avoid local optimization. Nevertheless, it is non-directional and hard to control.

In general, the searching process of typical chaos-based optimization can be divided into two stages. In the first stage, a bunch of chaotic sequences with certain length are generated by logistic chaos generating function. Then one or more gene-bits of individuals are changed according to the chaotic sequences and a series of new individuals are generated. After the selection of good solution among these new individuals, the second stage will introduce a small disturbance to the local optimum individuals for further exploitation. The iteration will continue until the terminate standards are satisfied.

However, two problems come up to restrain the performance of chaos for large-scale problems with irregular solution spaces. First, small disturbance will not help to exploit in complex and irregular spaces. Besides, the length of chaotic sequence directly decides the time consumption and searching ability of the algorithm.

For higher searching ability, the second problem is that fixed length of chaotic sequences may bring large time consumption in exploration. Thus, we design a new individual chaos operator in which the small disturbance is abandoned and adaptation of chaotic length is introduced for individuals with customization.

Specifically, the length of chaotic sequence for each individual is determined by its current evolutionary state. Let $\mathbf{I} = \{I_i | i = 1, 2, N\}$ be the population with N individuals, and I_i be the ith individual. It includes its gene-bit values $G_i = \{G_i(1), G_i(2), \ldots, G_i(M)\}$ and fitness value F_i, where M represents the length of gene code (i.e. twice of the number of tasks). The specific pseudo-code is shown as Algorithm 2.

Algorithm 2: Individual Chaos Operator

Individual_Chaos (**I**)
For ($i = 1$ to N)
$L_{chaos} = A + B(F_{best} - F_i) / (F_{best} - F_{worst} + 1)$
Generate $X_1[L_{chaos}]$, $X_2[L_{chaos}] \in [0,1]$ by using Logistic chaos function
For ($j = 1$ to L_{chaos})
Map $X_1(j)$ as genebit serial number $k \in [1, M]$
If (k corresponds to service-bit)
Map $X_2(j)$ as genebit value $v \in [1, n_s]$
Else
Map $X_2(j)$ as genebit value $v \in [1, n_p]$
End if
Generate j new temporary individuals $\{r(1), r(2), \cdots, r(j)\}$ by replacing the value of $G_i(k)$ with v
Choose the best individual r_{best} from the temporary individuals
If ($F_{r_{best}} > F_i$)
$I_i = r_{best}$
Else
If ($\exp((F_{r_{best}} - F_i) / t^o) > \gamma$)
Replace I_i with r_{best}
End if
End if
End for
$t^o = Dt^o$
End for

Go in detail, the evolutionary state of the ith individual is defined as Q_i:

$$Q_i = \frac{F_{best} - F_i}{F_{best} - F_{worst}} \tag{9.24}$$

$$L_{chaos} = A + (B - A)Q \tag{9.25}$$

where A and B is the lower bound and upper bound of L_{chaos}, respectively. F_{best} and F_{worst} represent the serial numbers of the individual with the best and the worst fitness value. To be exact, the closer the average fitness value to the best fitness value in population, the better the evolutionary state is, and the shorter the length

of chaotic sequence L_{chaos} is, so that the smaller the searching range is. Otherwise, the closer the average fitness value to the worst fitness value in population, the smaller the searching range is.

With the initialized definition of the length of chaotic sequences L_{chaos}, the operator generates two chaotic sequences $X_1[L_{chaos}]$, $X_2[L_{chaos}]$ for each individual $I_i(i = 1, 2, \ldots, N)$ by Logistic mapping chaotic function, as shown in Eq. (9.26).

$$z_{l+1} = \mu z_l(1 - z_l) \tag{9.26}$$

where $\mu = 4$ according to general chaotic strategy. Then X_1 and X_2 are mapped to the serial number k and the value v of gene-bits respectively. If $k \in [1, M]$ corresponds to service gene-bit, we should map X_2 to relative service number and store it in v. Or we should map X_2 to relative computing resource number and store it. In the pseudo-code, n_s and n_p represents the number of cloud services and the number of computing resources respectively. After the chaotic mapping step, new neighbor solutions $\{r(1), r(2), \ldots, r(j)\}$ can be generated by changing the value of $G_i(j)$ into v. Further, choose the individual r_{best} with the best fitness value and accept it as new individual with probability $P_{annealing} = \exp(\frac{F_{r_{best}} - F_i}{t^0})$, where t^0 is the annealing temperature and the initial value is 100. In the algorithm, the rate of t^0 drop D is set to be 0.95 to gradually narrow down the accept probability. On the whole, in the individual chaos operator, searching is carried out with the adaptive changing of the length of chaotic sequences for each individual according to its evolutionary state Q_i. Chaos states can finally be controlled by population state.

9.5.4 Dynamic Heuristic Operator

For further improving the searching direction in chaos optimization, dynamic heuristic is introduced in this algorithm after ranking selection and adaptive individual chaos. The principle of this operator is dynamically guiding the algorithm for local search with right direction by using some priori knowledge of the problem.

To be specific, for each individual, the operator randomly chooses a gene-bit, traverses part of the available values for the single gene-bit and dynamically calculates the heuristic of each value, then picks the most suitable value with the highest heuristic and generates new individual. It's quite like the mechanism of pheromone in ACO. Compared with the pheromone, dynamic heuristic here does not contain empirical information. It uses just the priori knowledge which is dynamically calculated according to the states or the gene-bit values of individuals in each generation. Define the traverse range for one gene-bit to be hn, where $h \in [0, 1]$, and n can be n_s or n_p. The specific pseudo-code is shown as follow.

Algorithm 3: Dynamic Heuristic Operator
Dynamic_Heuristic (**I**)
For ($i = 1$ to N)
$r = I_i$
Randomly choose a genebit $k \in [1:M]$
If (p corresponds to service-bit)
Randomly choose hn_s values from 1 to n_s
Choose the service s_i with the highest heuristic $Y_s = \max_j y_s(j)\ (j \in [1, hn_s])$
$G_r(k) = s_i$
Else
Randomly choose hn_p values from 1 to n_s
Choose the computing resource p_i with the highest heuristic $Y_p = \max_j y_p(j)\ (j \in [1, hn_p])$
$G_r(k) = p_i$
End if
Accept r with simulation annealing probability
End for

During the process, h can be set as 0.3. And p represents the randomly selected gene-bit for each individual. If p corresponds to service-genebit, search available services and calculate dynamic heuristic of each available service $y_s(j)(j \in [1, hn_s])$ by service heuristic function. Then choose the service s_i with the highest heuristic Y_s to replace the original value of kth gene-bit. If k corresponds to computing resource gene-bit, search available computing resources and calculate dynamic heuristic of each computing resource $y_p(j)(j \in [1, hn_p])$ by computing resource heuristic function. Then choose the computing resource p_i with the highest heuristic Y_p to replace the value of kth gene-bit. After these steps, a new individual r is generated for each individual. Then replace I_i with r in simulation annealing probability as well as in adaptive chaos operator $P_{annealing} = \exp(\frac{F_r - F_i}{t^0})$.

In this process, how to design service heuristic function and computing resource heuristic function is very important. Unsuitable heuristic function can cause wrong searching direction in algorithm and easily lead the algorithm to serious premature convergence. In this chapter, the service and computing resource heuristic function are simply designed as follow.

$$y_s(j) = \begin{cases} \alpha_1 \dfrac{s_{s_j}}{v_{s_j}} + \alpha_2 \dfrac{1}{r_{s_j}} + \alpha_3 \dfrac{1}{e_{s_j}} + \alpha_4 \dfrac{1}{c_{s_j}} + \alpha_5 r_{s_j}, & if\ s_j \in S_1 \\ \alpha_1 \dfrac{s_{s_j}}{v_{s_j}} + \alpha_2 \dfrac{1}{c_{s_j}} + \alpha_3 r_{s_j}, & if\ s_j \in S_2 \end{cases} \tag{9.27}$$

$$y_p(j) = \alpha_1 \frac{x(j)}{} \lambda(j) + \alpha_2 \max(\varphi(j), \phi(j)) + \alpha_3 \frac{1}{\sigma(j)} + \alpha_4 \frac{1}{f(j)} \tag{9.28}$$

where α_1, α_2, α_3, α_4, α_5 represent the weights of services/computing resources attributes respectively which corresponds to the weights setting in the objective function. Through the adjustment of weights, small range of local search in a single gene-bit could be guided in the algorithm according to the dynamic heuristics.

9.5.5 The Complexity of the Proposed Algorithm

Generally, the time complexity of the intelligent algorithms is dynamically varied with different problems. Let n be the scale of the population, m be the size of tasks, s be the number of the available cloud services for each task and p be the total scale of computing resources. The algorithms' complexities in each generation are shown in Table 9.2.

In GA, typical roulette wheel selection needs n times roulette operations to generating new population. Each roulette operation contains at least 1 and at most n times comparison according to the relative fitness values of individuals. Thus the average complexity of selection operator is $O(n^2)$. In RCO, the complexity of ranking individuals in selection is $O(n\log n)$ (with quick sort method) and the selection step according to selective pressure needs at most n times. Thus the complexity of ranking selection is $O(n\log n)$.

Besides, crossover and mutation operation in GA are just executed once for each individual. The complexity are both at least $O(n)$ and at most $O(mn)$. In RCO, chaotic sequences with constant length are generated for each individual. From the pseudo-code it can be seen that the complexity of individual chaos operator is $O(nL_{chaos}) = O(n)$. Because the adaptation of chaotic length is in a limited area, the complexity of chaos operator is also $O(n)$. It is lower than crossover operator. In addition, dynamic heuristic randomly chooses a gene-bit for each individual, traverse part of available value of this gene-bit with heuristics. If all of the selected gene-bits are service-bit, then the complexity is $O(n_s)$, else if all of the selected gene-bits are computing resource-bit, then the complexity is $O(n_p)$. Thus the average complexity of dynamic heuristic operator is $O(n(s + p)/2) = O(n\max(p, s))$.

In theory, if $s \to \infty$ and $p \to \infty$, the complexity of RCO is a little higher than GA. But in the condition of $n \to \infty$ and $m \to \infty$, the complexity of RCO is lower than GA.

9.6 Experiments and Discussions

Based on the case "the design and NC (Numerical Control) machining process of a complex surface part" mentioned before, three typical DAG: two DAGs as shown in Fig. 9.5 [21] and the "j30" DAG of Resource-Constrained Project Scheduling Problem (RCPSP) in PSPLIB [31], are used as three task graphs in our experiments. In practical application of private cloud in manufacturing conglomerate or large-scale manufacturing service providers, a composite project contains multiple complex surface parts' machining. Thus a composite project can be divided into far more than 5 tasks. Those tasks have several functional and non-functional

Table 9.2 The complexity of the operators in GA and RCO

Algorithms	The time complexities of operators			$n \to \infty$	$m \to \infty$	$s \to \infty$	$p \to \infty$
GA	Roulette wheel Selection	Crossover	Mutation	$O(n^2)$	$O(m)$	$O(1)$	$O(1)$
	$O(n^2)$	$O(nm)$	$O(nm)$				
RCO	Ranking Selection	Individual Chaos	Dynamic Heuristic	$O(n\log n)$	$O(1)$	$O(s)$	$O(p)$
	$O(n\log n)$	$O(n)$	$O(n*(\max(p, s)))$				

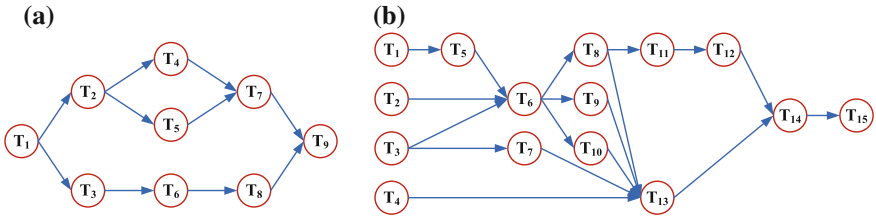


Fig. 9.5 Two typical DAG with 9 and 15 tasks respectively. **a** DAG1 **b** DAG2

requirements for cloud services. Some of them need hardware cloud services, some need software cloud services. In order to evaluate the performance of dual-scheduling optimization compared with the traditional two-level decision, we use basic real-coding GA uniformly to simulate the decision process in theory. At the OACR step, each gene-bit represents the selected computing resource number for the above selected service. The lengths of gene-bits at the two steps are equal. Furthermore, At the SCOS step, we consider only the properties of cloud services and then set the objective function as following according to Eq. (9.14) and [32].

At the OACR step, we also use the objective function in Eq. (9.14) with the fixed properties of cloud services. In Eq. (9.18), let the weight to be $w_1 = w_2 = w_3 = w_4 = 100n$.

For simplifying the optimization process, we set that each task in a composite project has the same number of available cloud services. In the three cases, 3 composite scales of DS-CSCR are tested, as shown in Table 9.3. And in each scales, computing resources are equally divided into 5 distributed groups.

Assume the available number of cloud services for each task is s and the available number of computing resources is p, then the size of solution space is $s^n p^n$. From Scale 1 to Scale 9, it's range from $10^9 \times 20^9$ to $50^9 \times 100^9$. Most deterministic algorithms can't handle these situations due to composite exposition.

Table 9.3 The selected 4 composite scales of cloud services and computing resources

	Scale 1	Scale 2	Scale 3	Scale 4	Scale 5	Scale 6	Scale 7	Scale 8	Scale 9
Number of tasks	9	9	9	15	15	15	30	30	30
Number of available cloud services	10	20	50	10	20	50	10	20	50
Number of available computing resources	20	50	100	20	50	100	20	50	100

Table 9.4 The property ranges of cloud services and computing resources

	s	c	r	v	e	ζ
Software service	[1, 10]	[1, 10]	(0, 1)	[1, 10]		
Hardware service	[1, 10]	[1, 10]	(0, 1)	[1, 10]	[1, 10]	[0, 1]
.	x	φ	ϕ	σ	f	λ
computing resource	[1, 10]	[1, 10]	[1, 5]	[1, 10]	(0, 1)	0

Moreover, because of the restriction of experimental environment, we set the ranges of properties of cloud services and computing resources as shown in Table 9.4.

For theoretical analysis, all the values are randomly generated with normalization and idealization and stored in a *txt* file. In order to distinguish the bandwidths inter-group and intra-group, the range of φ is set to be slightly larger than ϕ. Initially, task load of all computing resources are 0.

Based on DS-CSCR with 9 scales, standard GA, chaos GA (CGA), typical chaos optimization (CO), chaos optimization with only individual chaos operator designed in this chapter (RCO^{-2}), chaos optimization with ranking selection and individual chaos operator (RCO^{-1}) and chaos optimization with the addition of dynamic heuristics (RCO) are compared together for further testing the performance of the above designed algorithm. In the experiments, the classical roulette wheel selection operator, multiple-point crossover operator and single-point mutation operator are adopted in GA. And the crossover and mutation probabilities are set to be the typical values, i.e. 0.8 and 0.15, respectively. In chaos strategy of CGA and CO, the length of chaotic sequences is set as a constant 10. For a fairer comparison, in the new RCO, let $A = 5$ and $B = 15$ to make sure the same level of chaotic operation. Besides, the iterations of all experiments are set as 2000 uniformly and population sizes are all 20. Due to the randomness of intelligent algorithms, a total of 100 runs of each experiment are conducted and the average fitness value of the best solutions throughout the run is recorded.

9.6.1 Performance of DS-CSCR Compared with Traditional Two-Level Scheduling

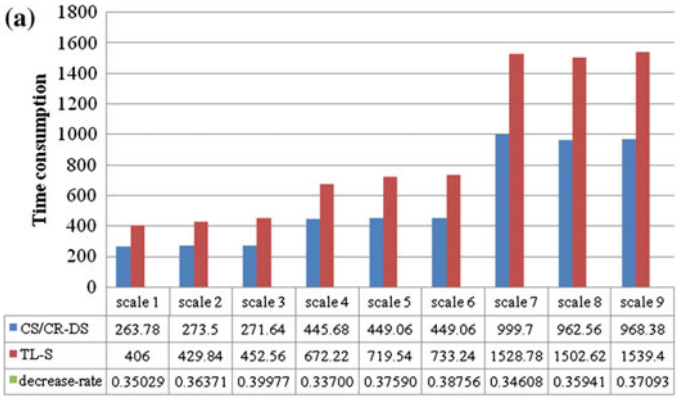
Let TL-S to be the abbreviation of traditional Two-Level Scheduling, we compared it with new DS-CSCR in the above 9 scales of solution space. Figure 9.6 shows the testing results from the perspectives of time consumption and solution quality respectively.

Firstly, we define the *decrease-rate* to be $\tau_d = \frac{T_{TL-S} - T_{CS/CR-DS}}{T_{TL-S}}$ in Fig. 9.6a. As we have analyzed previously, the time consumption of SCOS and OACR in traditional TL-S are reduced by about 35–40 % in DS-CSCR. Although the length of individual and the size of solution space are only half that of DS-CSCR. Traditional TL-S takes almost twice the time of DS-CSCR. For each task graph, as the numbers of cloud services and computing resources are enlarged, the *decrease-rate* increases gradually. Thus it can be seen, with the same algorithm (no matter deterministic algorithm or intelligent algorithm), TL-S is more and more time-consuming with the increase of solution space while DS-CSCR always maintains a relatively low level of time consumption. It proved that, with the same algorithm, no matter using determistic or intelligent, two level decision is cumbersome.

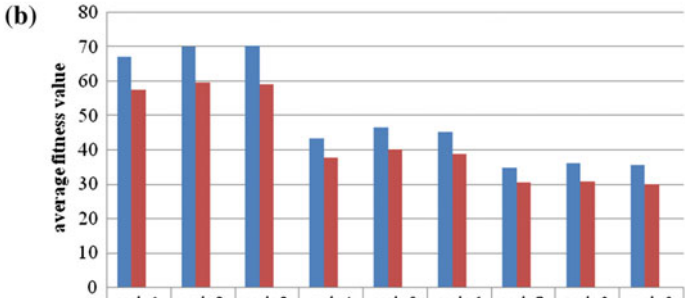
Secondly, from the angle of solution quality in Fig. 9.6b, we define the *growth-rate* to be $\tau_g = \frac{\bar{F}_{CS/CR-DS} - \bar{F}_{TL-S}}{\bar{F}_{TL-S}}$, where $\bar{F}_{CS/CR-DS}$ and $\bar{F}_{TL-S}$ represent the average result of the best fitness value in experiments. It increases along with the expansion of solution spaces in each kinds of task graph. For all of scales, the total level of quality in TL-S is improved by about 14–19 % in DS-CSCR. In theory, the service properties are static in the second step of TL-S. With the splitting of SCOS and OACR under unified console, the mutual relations between cloud service and the underlying computing resources are ignored. This tells us the conclusion that in private cloud, the underlying support infrastructure must be considered in the process of SCOS. With fast development of dynamic network, service with dynamic deployment are more and more common. SCOS with the consideration of QoS only are inpractical for many of the advanced system in large SaaS mode.

9.6.2 Searching Capability of RCO for Solving DS-CSCR

For addressing DS-CSCR more efficiently, we designed RCO especially aiming at the situation of large-scale solution space. Figure 9.7 recorded the average fitness value of the best solution during 2000 generations in 100 runs for 9 scales of DS-CSCR (i.e. the average evolutionary trend of the 6 algorithms in 100 runs). Figure 9.8 shows the fitness value of the best solution, the worst solution and the average result in 100 runs for 9 scales of DS-CSCR. Note that the fitness value is the assessment value of each individual according to the objective function. So from the perspective of searching capability, the sort of the six algorithms from



	scale 1	scale 2	scale 3	scale 4	scale 5	scale 6	scale 7	scale 8	scale 9
CS/CR-DS	263.78	273.5	271.64	445.68	449.06	449.06	999.7	962.56	968.38
TL-S	406	429.84	452.56	672.22	719.54	733.24	1528.78	1502.62	1539.4
decrease-rate	0.35029	0.36371	0.39977	0.33700	0.37590	0.38756	0.34608	0.35941	0.37093



	scale 1	scale 2	scale 3	scale 4	scale 5	scale 6	scale 7	scale 8	scale 9
CS/CR-DS	67.0273	70.0346	70.2574	43.2662	46.5191	45.2871	34.9133	36.1147	35.5721
TL-S	57.5089	59.7877	59.0841	37.7113	40.1413	38.6902	30.5586	30.8497	30.0211
growth-rate	0.16551	0.17138	0.18910	0.14730	0.15888	0.17050	0.14250	0.17066	0.18490

Fig. 9.6 Comparison of DS-CSCR and TL-S based on GA. **a** The average solution of DS-CSCR and TL-S based on GA in 9 scales **b** The average solution of DS-CSCR and TL-S based on GA in 9 scales

bad to good is: GA<CGA<CO<RCO^{-2}<RCO^{-1}<RCO. The step-by-step improvement from the design of individual chaos operator to the introduction of ranking selection and dynamic heuristic operators can be clearly observed.

On the basis of GA, the average fitness value of the best solutions of CGA is about 30 % higher than GA. At this moment, the average best fitness value of CO with single chaos optimal operator is about 1.5 times higher than GA. From here we can come to the conclusion that the basic operators of GA constrained the searching ability of chaos optimal operator in CGA to some degree. Simple chaos optimization can get much better solution than the traditional GA and improved CGA. Furthermore, the adaptive strategy adapts chaotic sequences according to the state of the whole population. When the population is in a good state, the adaptive strategy will reduce the chaotic sequences, so as to reduce the complexity of the algorithm. Compared with CO, the average best fitness value

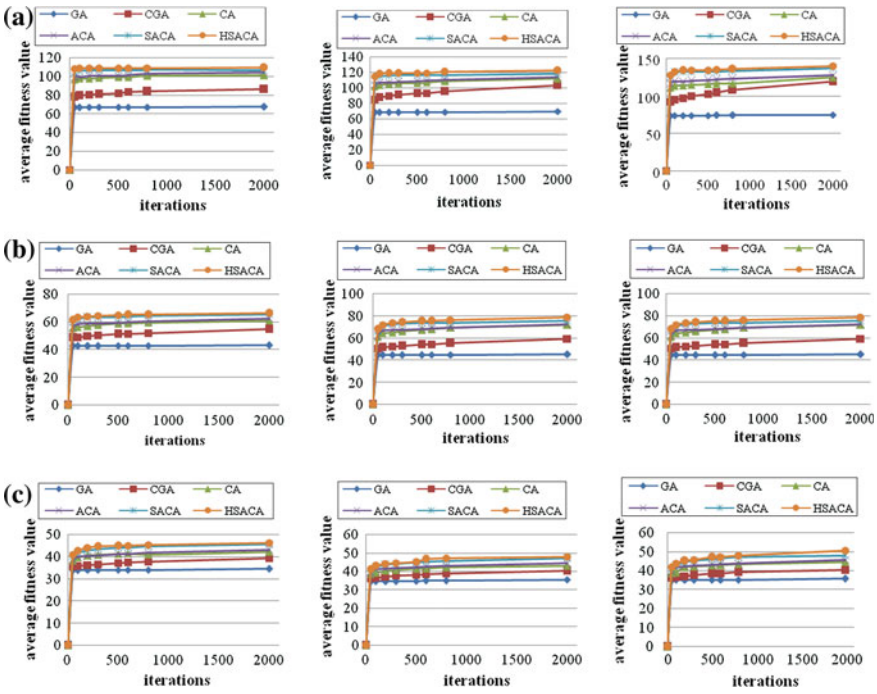


Fig. 9.7 The average evolutionary trend of the 6 algorithms in 100 runs for 9 scales of DS-CSCR. **a** DAG1 with 9 tasks in the scale 1, 2 and 3 **b** DAG2 with 15 tasks in the scale 4, 5 and 6 **c** DAG3 with 30 tasks in the scale 7, 8 and 9

of RCO^{-2} in the 9 scales of DS-CSCR has been raised by about 3 %. Afterwards, ranking selection was put in the front of RCO^{-2}. With the collaboration of selection and chaos, the average best fitness value of RCO^{-1} is improved again. Hence, it can be learned that the operation and collaboration of individual chaos operator and the "the survival of the fittest" ranking selection strategy can not only reduce the complexity of algorithm, but also improve the searching capability remarkably. Because the effect of mutation is similar to chaos operator, it may conclude that the crossover operator in GA mainly restrained the capability of chaos strategy in CGA. Based on the improved RCO^{-1}, for guiding chaos optimization further, dynamic heuristic operator was introduced at last. From Figs. 9.7 and 9.8 we can see that the new RCO performs better than RCO^{-1} with the guidance of heuristics. On the whole, the average best fitness value of RCO in 100 runs is about 2 times higher than GA. The overall improvements are extremely considerable.

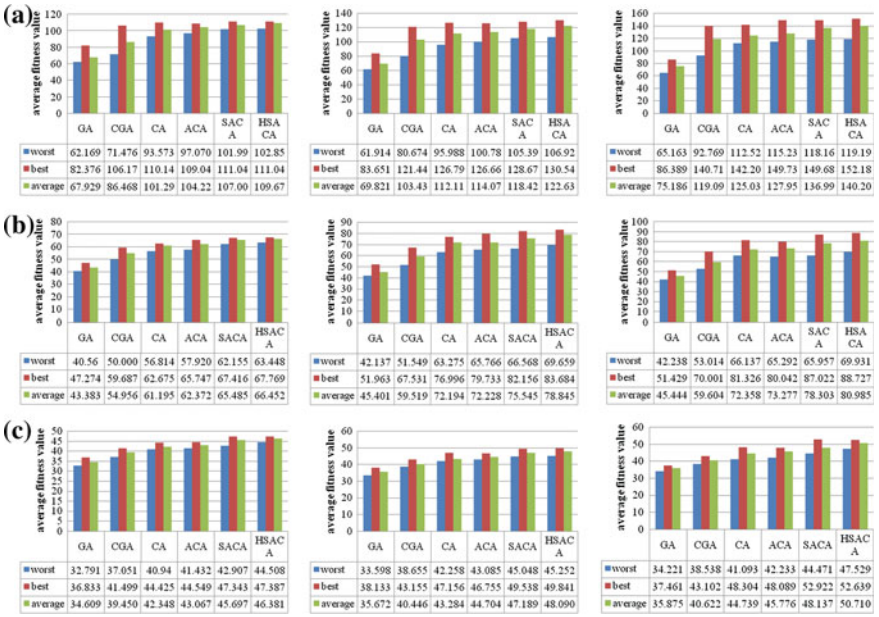


Fig. 9.8 The statistical results of the 6 algorithms in 100 runs for 9 scales of DS-CSCR. **a** DAG1 with 9 tasks in the scale 1, 2 and 3 **b** DAG2 with 15 tasks in the scale 4, 5 and 6 **c** DAG3 with 30 tasks in the scale 7, 8 and 9

9.6.3 Time Consumption and Stability of RCO for Solving DS-CSCR

Next, based on the above mentioned 9 scales with 3 kinds of task graphs (Table 9.3), the time efficiency and stability of the 6 algorithms are discussed below. Note that the time consumption are tested in millisecond (ms) and the stability is measured by the standard deviation of the average fitness values in 100 runs.

Figure 9.9a shows the average time-consumption of the 6 algorithms in 9 scales with 100 runs. The step-by-step improvement from CO to RCO compared with GA and CGA, the variation trends of time in all scales are the same. In CGA, there are four operators (selection, crossover, mutation and chaos), with lower searching capability, its time-consumption is the highest in these 6 algorithms. After wiping out the three operators of GA, the times of CO are just lower than CGA. It is clear that the most time-consuming operator in CGA is chaos operator. Only narrowing down the chaotic traverse range can reduce the total execution time of algorithm. Along with the decrease of chaotic sequences, the searching ability of algorithm will be reduced, too. Therefore, in order to reduce the time complexity of algorithm with the maintaining of the searching ability, individual chaos operator customized for individuals is designed in this chapter. Experiments in RCO^{-2}

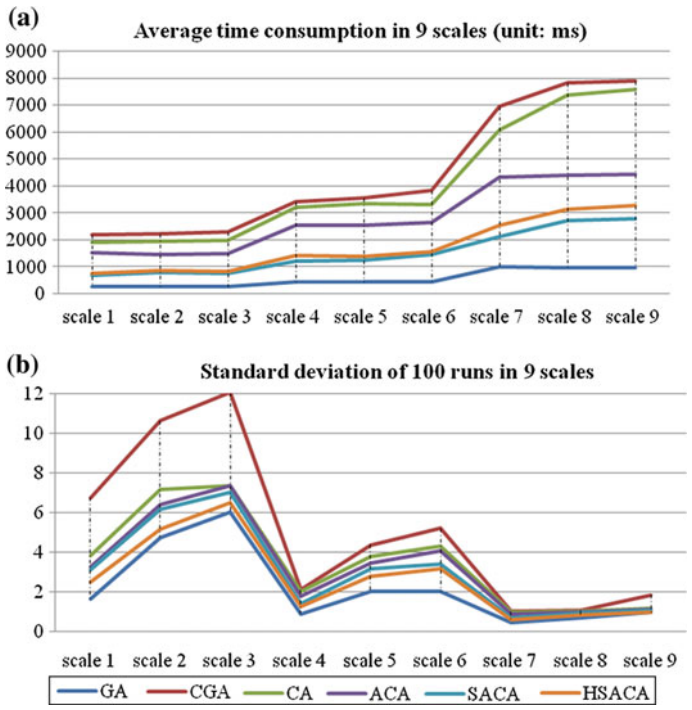


Fig. 9.9 The average time-consumption and standard deviation of the 6 algorithms in 100 runs. **a** Average time consumption of 6 algorithms **b** Standard deviation of 6 algorithms

show that the time-consuming is effectively reduced by about 20 % based on CO with the improvement of searching ability. Especially in scale 7, 8 and 9 with very large solution spaces, time-consuming of chaotic operations are sharply reduced.

Moreover, the introduction of ranking selection not only improved the searching capability of RCO^{-2}, but also reduced the time. The reason is that, based on ranking selection, the difference between the best fitness value and the average fitness value in the population is shortened, the population can always be adapted to a better state with "the survival of the fittest" strategy, then the chaotic sequences are shortened accordingly. With shorter chaotic sequences, the population can be guided to better areas based on fitter individuals and then find better solutions more quickly. In terms of the time measuring, the prominent performance of the collaborative operation of ranking selection and individual chaos operator has been verified again as RCO^{-1}. At the next step, the introduction of dynamic heuristic operator increase the time slightly based on RCO^{-1}, but the new complete RCO is much faster than RCO^{-2}, CGA and CO as a whole.

From the perspective of stability, as shown in Fig. 9.9b, the six algorithms in the 9 problem scales changed irregularly. But from the 9 scales of tests, we can obtain the sort of stability of the six algorithms from bad to good is:

CGA<CO<RCO^{-2}<RCO^{-1}<RCO<GA. Traditional GA is the most stable while the stability of CGA is the worst. With the adaptive improvement, RCO^{-2} is more stable than CO. That is because in large-scale solution space, chaotic sequences are generated based on no matter good or bad individuals, the population is easy to be lead to bad areas during searching and the states of population in each generations are not stable any more. After the introduction of ranking selection, the stability of the algorithm has greatly improved. Each time of selection in iteration maintained the population state and reduced the chaotic sequences, so that the population can always be evolved based on fitter individuals with higher stability. Besides, the design of dynamic heuristic operator with the priori knowledge of DS-CSCR can always guide the population into better areas during evolution and then improve the stability further.

Thus it can be seen that the new designed RCO possesses plenty of advantages in searching capability, time-consumption and stability for addressing DS-CSCR no matter with large or small scales solution spaces in private cloud.

9.7 Summary

Service composition optimal selection (SCOS) and optimal allocation of computing resource (OACR) are both very critical in cloud system. Current works found that the two steps decision of SCOS and OACR in private cloud are quite cumbersome and the mutual relations between cloud services and underlying computing resources are always ignored. Thus this chapter deeply analyzed the characteristics of these two problems and their interactions. Based on this, the idea of one-time decision of SCOS and OACR was presented accordingly. To sum up, the primary works of this chapter can be concluded as follows.

(1) New DS-CSCR model was presented in private cloud for high efficient one-time decision. Properties of software/hardware cloud services, VMs and computing resources are deeply analyzed. The formulation of DS-CSCR was clarified according to the aim of high efficient and low cost resource sharing.
(2) For addressing the complex dual scheduling problem (DS-CSCR), a new intelligent algorithm—RCO was presented. Individual chaos operator was designed as the backbone operator of the algorithm. Then a new adaptive ranking selection was introduced for control the state of population in iteration. Moreover, dynamic heuristics were also defined and introduced to guide the chaos optimization. RCO with these three operators showed remarkable performances in terms of searching ability, time complexity and stability in solving the DS-CSCR problem in such private cloud compared with other algorithms.

References

1. Laili YJ, Tao F, Zhang L, Cheng Y, Luo Y, Sarker BR (2013) A ranking chaos algorithm for dual scheduling of cloud service and computing resource in private cloud. Comput Ind 64(4):448–463
2. Boss G, Malladi P, Quan D, Legregni L, Hall H (2007) Cloud computing. IBM White Paper, 2007. http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud_computing_wp_final_8Oct.pdf
3. Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2009) Above the clouds: a berkeley view of cloud computing. University of California, Berkeley
4. Xia TZ, Li Z, Yu NH (2009) Research on cloud computing based on deep analysis to typical platforms. Lect Notes Comput Sci 5931:601–608
5. Xu X (2012) From cloud computing to cloud manufacturing. Robot Comput Integr Manuf 28(1):75–86
6. Wu D, Thames L, Rosen D, Schaefer D (2012) Towards a cloud-based design and manufacturing paradigm: looking backward, looking forward. In: Proceedings of the ASME 2012 international design engineering technical conference and computers and information in engineering conference, Chicago
7. Vaquero LM, Rodero-Merino L, Caceres J, Lindner M (2009) A break in the clouds: towards a cloud definition. ACM SIGCOMM Comput Commun Rev 39(1):50–55
8. Li BH, Zhang L, Wang SL, Tao F, Cao JW, Jiang XD, Song X, Chai D (2010) Cloud manufacturing: a new service-oriented networked manufacturing model. Comput Integr Manuf Syst 16(1):1–16
9. Nick JM, Cohen D, Kaliski BS (2010) Key enabling technologies for virtual private clouds. Handb Cloud Comput 1:47–63
10. Tan W, Fan YS, Zhou MC (2010) Data-driven service composition in enterprise SOA solution: a petri net approach. IEEE Trans Autom Sci Eng 7(3):686–694
11. Tao F, Hu YF, Zhao D, Zhou ZD, Zhang HJ, Lei ZZ (2009a) Study on manufacturing grid resource service QoS modeling and evaluation. Int J Adv Manuf Technol 41 (9-10):1034–1042
12. Tao F, Hu YF, Zhou ZD (2009b) Application and modeling of resource service trust-QoS evaluation in manufacturing grid system. Int J Prod Res 47(6):1521–1550
13. Tao F, Zhao D, Hu YF, Zhou ZD (2010) Correlation-aware resource service composition and optimal-selection in manufacturing grid. Eur J Oper Res 201(1):129–143
14. Fujii K, Suda T (2005) Semantics-based dynamic service composition. IEEE J Sel Areas Commun 23(12):2361–2372
15. Ferrer AJ, Hernandez F, Tordsson J, Elmroth E, Ali-Eldin A, Zsigri C, Sirvent R, Guitart J, Djemame RM, Ziegler W, Dimitrakos T, Nair SK, Kousiouris G, Konstanteli K, Varvarigou T, Hudzia B, Kipp A, Wesner S, Corrales M, Forgo N, Sharif T, Sheridan C (2012) OPTIMIS: a holistic approach to cloud service provisioning. Future Gener Comput Syst 28(1):66–77
16. Mika M, Waligora G, Weglarz J (2011) Modeling and solving grid resource allocation problem with network resources for workflow applications. J Sched 14(3):291–306
17. Tordsson J, Montero RS, Moreno-Vozmediano R, Liorente IM (2012) Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Gener Comput Syst 28(2):358–367
18. Endo PT, Palhares AVD, Pereira NN, Goncalves GE (2011) Resource allocation for distributed cloud: concepts and research challenges. IEEE Netw 25(4):42–46
19. Ma YB, Jang SH, Lee JS (2011) QoS and ontology-based resource management in cloud computing environment. Inf Int Interdisc J 14(11):3707–3715
20. Xiong PC, Chi Y, Zhu SH, Moon HJ, Pu C, Hacigumus H (2011) Intelligent management of virtualized resources for database systems in cloud environment. In: Proceedings of the 27th IEEE international conference on data engineering

21. Zhang YH, Li YH, Zheng WM (2011) Automatic software deployment using user-level virtualization for cloud-computing. Future Gener Comput Syst 29(1):323–329
22. Ghanbari H, Simmons B, Litoiu M, Iszlai G (2012) Feedback-based optimization of a private cloud. Future Gener Comput Syst 28(1):104–111
23. Laili YJ, Tao F, Zhang L, Sarker BR (2012) A study of optimal allocation of computing resources in cloud manufacturing systems. Int J Adv Manuf Technol 63(5–8):671–690
24. Nathani A, Chaudhary S, Somani G (2012) Policy based resource allocation in IaaS cloud. Future Gener Comput Syst 28(1):94–103
25. Ma Y, Zhang CW (2008) Quick convergence of genetic algorithm for QoS-driven web service selection. Comput Netw 52(5):1093–1104
26. Yin PY, Wang JY (2008) Optimal multiple-objective resource allocation using hybrid particle swarm optimization and adaptive resource bounds technique. J Comput Appl Math 216(1):73–86
27. Wada H, Suzuki J, Yamano Y, Oba K (2011) Evolutionary deployment optimization for service-oriented clouds. Softw Pract Exp 41(5):469–493
28. Tao F, Zhang L, Venkatesh VC, Luo YL, Cheng Y (2011) Cloud manufacturing: a computing and service-oriented manufacturing model. Proc Inst Mech Eng Part B J Eng Manuf 225(10): 1969–1976
29. Schaefer D, Thames L, Wellman RD, Wu D (2012) Distributed collaborative design and manufacture in the cloud–motivation, infrastructure and education. In: Proceedings of the annual conference and exposition (ASEE), Texas
30. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comput Syst 28(5):755–768
31. Kolisch R, Sprecher A (1997) PSPLIB-a project scheduling problem library: OR software-ORSEP operations research software exchange program. Eur J Oper Res 96(1):205–216
32. Tao F, Zhao DM, Hu YF, Zhou ZD (2008) Resource service composition and its optimal-selection based on swarm optimization in manufacturing grid system. IEEE Trans Ind Inf 4(4):315–327

Part V
Application of Parallel Intelligent Optimization Algorithms

Chapter 10
Computing Resource Allocation with PEADGA

In this chapter, for solving optimal allocation of computing resources (OACR) problem in cloud manufacturing (CMfg) [1], serial three-layer operation configuration and parallel configuration are both applied. Firstly, A new comprehensive model for OACR is proposed in CMfg system. In this model, all main computation, communication and reliability constraints in the special circumstances are considered. Secondly, niche strategy, immune heuristics, genetic operators and pheromone strategy are configured together to generate a hybrid niche immune algorithm (NIA) [2]. Based on NIA, we introduce an adaptive full mesh exchange scheme with population supervision and get a new parallel NIA (PNIA) for addressing the specific problem. From the perspective of algorithm parallelization, the supervision of population state is encapsulated as a module used before topology-based communication as an execution condition. Then the new module is configured together with full mesh topology in different generation.

10.1 Introduction

Nowadays, in the development of manufacturing, informatization is important. It connects enterprises to work together, share resources and improve the product efficiency. To fulfill the target of agility, high performance and low cost among enterprises all over the world, many manufacturing informatization modes, for example, agile manufacturing (AM) [3], application service provider (ASP) [4] and manufacturing grid (MGrid) [5] and so on, are proposed and used widely. Most of them are emphasis just on how to connect distributed resources by network with less considering of resource management and generalized dynamic sharing. At the same time, cloud computing as a new network application mode is springing up. It constructs computing service center and hire the computing power and storage by using virtualization technology. It combines multiple computing

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_10

resources and information as a strong "cloud" and divides computing power and storage quickly and freely from cloud to user on-demand through network. Cloud is just like a huge repository (and management) of resources which reflects the generalized dynamic sharing and cooperative management of resources.

Inspired by this, Cloud Manufacturing (CMfg) was presented by Li et al. [1] to expand the service mode in manufacturing informatization and improve its dynamic. It is a new networked manufacturing mode which aims at achieving low cost resource sharing and effective coordination. It transforms all kinds of manufacturing, simulation and computing resources and abilities into manufacturing services to form a huge "manufacturing cloud" and distributes them to user on-demand. In CMfg, there's a platform which combines core technologies of cloud computing, internet of things (IoT) and high performance computing (HPC) and so on to implement the intelligent management, efficient collaboration and dynamic arbitrary service composition and division. All these resources and abilities are intelligently sensed and interconnected into "cloud" and automatically managed via Internet to execute various manufacturing tasks [6, 12, 13]. That is to say, in manufacturing process, CMfg platform can analyze and divide users' requests and automatically search suitable information, available manufacturing devices and computing resources and intelligently integrate and provide them to users. Users here can hire remote large equipments and computing resources without buying, get more specific information about design, simulation, production, delivery and recycle and monitor the whole task execution process. Thus, the whole life cycle manufacturing process in CMfg can be simplified in Fig. 10.1. With high intelligence and information, it is a high level extension of service-oriented manufacturing and cloud computing.

Based on this idea, people would ask, how to transform large devices as services for hiring, how to implement efficient resources allocation and integration? Actually, they are all supported by computing resources, as shown in Fig. 10.1. Computing Resources, including CPU, processor, I/O, at the physical layer [1] is the core infrastructure of CMfg platform. They not only provide computing power as in cloud computing, but also control a variety of other manufacturing resources and abilities directly for collaboration and sharing. They locate in different places and form a big resource pool in CMfg platform through virtualization. Information sharing needs them, manufacturing devices invoking needs them and computing/simulation work needs them, too.

In other word, under the centralized management, various heterogeneous computing resources are integrated and re-divided as virtual machines by virtualization and assigned to user on-demand for computing and simulation. Meanwhile, when manufacturing equipments access in CMfg platform through transducers, computing resources then become a kind of control and management media. They encapsulate and map these manufacturing equipments as virtual resources with virtualization technology to support the effective interoperation, collaboration and monitoring of manufacturing tasks [7, 55]. High virtualization of all kinds of manufacturing hardware/software resources and high heterogeneity

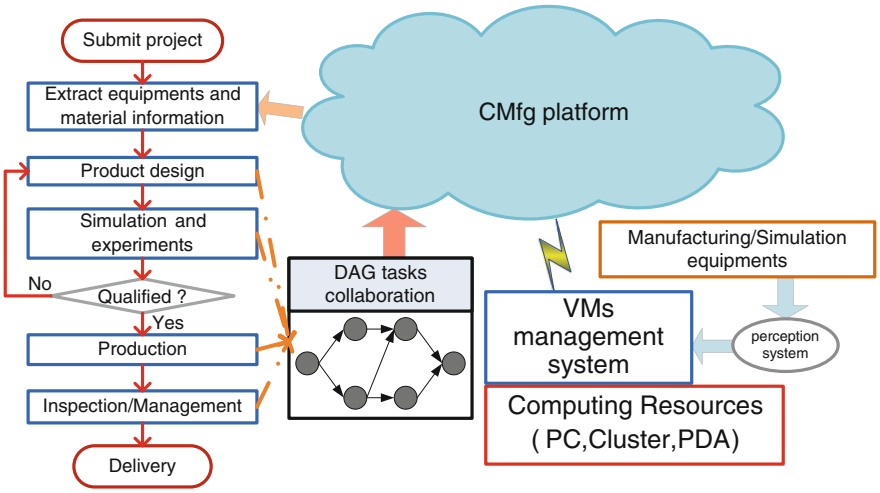


Fig. 10.1 The simplified manufacturing process in CMfg

and distribution of computing resources are two key characteristics of CMfg compared with cloud computing. Therefore, the optimal allocation of computing resources (OACR) which means efficient dividing and scheduling computing resources in manufacturing process for full utilization and high efficient operation is one of the most primary problems in CMfg.

Besides, oriented to the whole manufacturing life cycle, manufacturing tasks are very complex. They usually include multi-disciplinary collaborative tasks such as mechanical, electronic or control simulation and manufacturing. The demands of tasks for communication and computation power of manufacturing resources are high and different. Unlike the previous scheduling problems [8, 9] in parallel computing systems, in CMfg, computing resources are divided into virtual machines and allocated to different tasks according users' requirements. It has the characteristics of large scale, high heterogeneity, dynamic interconnection and group collaboration, which has imposed a new challenge on the construction of CMfg platform.

So, focusing on OACR problem in CMfg, we proposed a systematic model for it from the point of packet communication and partition of computing power. The detailed running process of the allocation of computing resources for manufacturing tasks is shown. Classical intelligent algorithms are introduced and compared in solving the problem, and a new improved hybrid intelligent algorithm, NIA, is configured to solve OACR. Further, a new topology with pre-handling module is configured and applied in NIA. Simulation results on standard tests show that this new algorithm is pretty efficient to solve this kind of high dimensional complex problems.

10.2 Related Works

To perform larger-scale collaborative manufacturing, CMfg was firstly presented by Li et al. [1]. They specifically defined it and introduced the architecture of CMfg. Based on this, many studies about CMfg are started. Zhang et al. [7, 55] further described the key technologies for the construction of CMfg. He defined the dynamic cloud services center in CMfg as manufacturing cloud and classified it as public cloud and private cloud. Then, from the perspective of the structure of manufacturing cloud, he elaborated the types of manufacturing resources, the dynamic sensing and accessing of hardware/software and the method of information exchange in CMfg. And for further understanding and the research of CMfg, Zhang [10] then analyzed the differences and connections among CMfg and other related advanced manufacturing modes and then presented the target of CMfg, i.e., agility, servicesation, greening and intelligent in the whole manufacturing. Based on these researches, Li et al. [11, 49] specified the characteristics of CMfg and presented argument as a service (AaaS), design as a service (DaaS), fabrication as a service (FaaS), experiment as a service (EaaS), simulation as a service (SimaaS), management as a service (MaaS) and integration as a service (InaaS). These concepts is inspired by cloud computing but clearly distinguished CMfg from cloud computing. At the same time, Tao et al. [6, 12, 13] elaborated the operational process of cloud manufacturing, the relation among resources, cloud service and cloud platform and the importance of optimal allocation of whole manufacturing resources and tasks in CMfg. All of these studies are macro-researches with less micro-analysis in each key part. However, in detail, how to implement intelligent and agility in optimal allocation of computing resources for supporting these advanced manufacturing process, as one of the most important thing of constructing CMfg platform, still hasn't been studied.

In manufacturing system, job-shop scheduling and workflow scheduling are much popular [61, 62] while the allocation of computing resources considered little. But from the global perspective, OACR is one of the most basic and important problem. OACR is a kind of pre-scheduling problem. It's more complex than several kinds of traditional job-scheduling or task scheduling problems [14–16]. In the existing task scheduling models, tasks can usually be expressed in four types: DAG (Directed Acyclic Graph) [17], HTG (Hierarchical Task Graph) [18], TIG (Task Interaction Graph) [19, 20] and Petri net [21]. The most commonly used is DAG, in which the nodes represent individual tasks and the directed arcs stand for communication overhead between tasks [22, 23]. Early DAG models were simplified as: the execution time of tasks are all the same, communication between tasks are excluding, the intercommunication interfaces between processors are enough and multiple communications can be performed simultaneously [17], and so on. The traditional DAG task scheduling problems have been proven to be NP-Complete [9]. It's far more complex in many kinds of manufacturing systems [1, 52, 63–65]. About the attributes of tasks, the concept of similarity is often

expressed as Granularity [24, 25], which indicates the ratio of communication overhead in a parallel program. The amount of communication edges is usually expressed as DAG density [26]. Besides, a variety of QoS (Quality of Service) indexes were also introduced in DAG, particularly in manufacturing task scheduling. Based on these QoS (Quality of Service) indexes, existing researches primarily focus on homogeneous cluster systems [27–29] scheduling more thread level tasks to less processors. The most frequently used topologies of the parallel systems are full interconnected network, hypercube network, grid network, public bus network, and so on [30]. The studies about heterogeneous systems are seldom. Typically, end-point and network communication contention in heterogeneous systems are analyzed by O Sinnen [31]. The communication preparation, overhead and involvement of processors and communication mode of task scheduling are elaborated by Sinnen et al. [32] and Benoit et al. [33], and so on.

On the scheduling algorithms side, typical deterministic algorithms are list scheduling [34–36], clustering scheduling [24, 37, 38], linear programming [39], stochastic mapping [40], and several others. Yu-Kwong and Ishfaq compared and summarized 15 types of scheduling algorithms in [17], which is widely cited. After that, a few efficient approximate algorithms [41, 42], were presented for solving these problems in acceptable times. With the increase of tasks and processors scale, traditional deterministic algorithms and original approximate algorithms can no longer meet the demand. Thus intelligent algorithms, such as genetic algorithms (GA) [43–46], ant colony optimization (ACO) [11, 47–49], immune algorithms (IA) [50, 51] and so on and other new heuristic approaches [52–54, 64] have been paid attention and widely applied to this kind of scheduling problems for finding the Pareto optimal solutions especially in manufacturing application field [66, 67].

However, the above-mentioned models are not practicable to CMfg. First, unlike the previous thread level tasks, manufacturing tasks (MTs) are usually carried by virtual machines (VMs) [1]. Virtual machines not only execute high performance computing tasks, but also supervise and control manufacturing hardware resources such as simulation equipment and machine tools. Users have different demands on them. Multi VMs can run in same processor. The more VMs carried at one processor, the slower their run. More importantly, there are frequent interactions between users and VMs during tasks' execution. In the other word, VMs generally execute coarse grain manufacturing tasks. Second, computing resources (CRs) with high heterogeneity are composed of different kinds of cluster, PC, PDA, and so on. They are scattered around the world with dynamic access, so the system topology is dynamic and uncertain. Hence different areas have different access bandwidths, links and communication buffers [7, 55]. Third, on CMfg platform, CRs have larger scale while MTs have relatively smaller scale with higher and complex demands. Based on such a complex system, therefore, OACR is different from the original scheduling problems and a detailed analysis of its new model and algorithms are presented in this chapter.

10.3 Motivation Example of OACR

A CMfg system consists of manufacturing resources, manufacturing cloud (CMfg platform) and the whole lifecycle manufacturing applications. Like the traditional service-oriented manufacturing modes, three user types – resource providers, cloud operators and resource users are included in the platform, as shown in Fig. 10.2 [7, 55]. Manufacturing cloud senses and manages the manufacturing resources (hardware/software) from resource providers all over the world. When users submit a manufacturing mission to manufacturing cloud, the platform analyzes the mission and intelligently divides it into sub-tasks in accordance with the requirement number of VMs and devices and then forms them as a DAG. That means each sub-task in DAG can be executed by only one VM or one device without separation. After the task partition, manufacturing cloud need to find available resources for each sub-task and provide them as services for users. In fact, as introduced in Sect. 10.1, all of the interactive and run processes among them are not only supported by knowledge, but also by computing resources.

In order to show the importance of OACR among the triple process, we specific the abstract workflow of task execution in CMfg as shown in Fig. 10.3 and consider the multi-disciplinary physical collaborative simulation for example.

Normally, for an accurate design and modeling in industrial manufacturing (such as airplane and automobile), physical collaborative simulation is important. On one hand, it needs collaborative simulation of Matlab and Adams and so on. On the other hand, it also needs driving simulator, multi-axis table and visual equipment to work together along with software. So it is a complex process in manufacturing. Assume there is a physical simulation task submitted to manufacturing cloud. After a series of intelligent divisions of task, the following steps are done in the CMfg platform.

(1) *Requirement analysis of task DAG*: According to the users' requirement of DAG, analyze the communication and computation costs and the QoS (Quality of Services) constraints of tasks. Then check the accessed resources(include computing resources and manufacturing devices). If there's no available resource or the resources are not enough, then reject the tasks. Or the system will send a confirmation message to users and then take the next step.

(2) *Optimal allocation and strategy sending*: In terms of the QoS and costs of tasks, manufacturing cloud determines which tasks need remote simulation physical devices. If the task needs physical device, then calculates the attribute values of device and maps it to the requirement attributes of controlling VM. Else the platform only needs to calculate the requirement attributes of computing VM for task. As soon as the platform establishes these VMs' requirement, it executes a scheduling algorithm for mapping these VMs to available computing resources, then gets the optimal allocation of computing resources strategy and sends it to users.

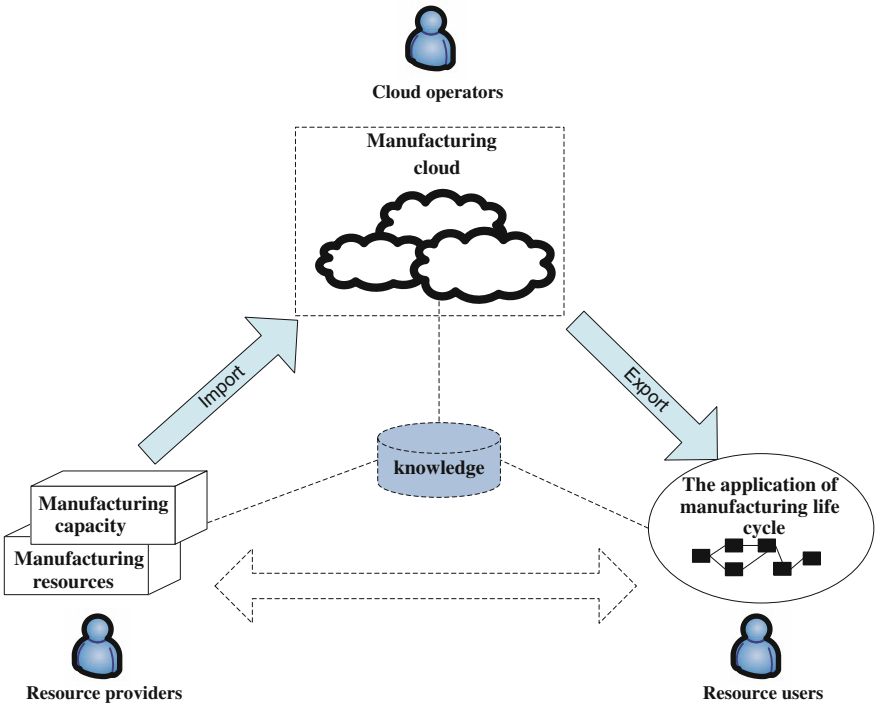


Fig. 10.2 The abstract operation principle of CMfg

(3) *Execution*: After the users' confirmation, manufacturing cloud then invokes these VMs and simulation hardware to execute. The simulation runtime process could be controlled and monitored by users through controlling VMs on Internet. If unexpected error occurs during execution, the platform will call the fault-tolerant migration strategy automatically and try to execute tasks again.

(4) *Result receiving and resources release*: At the end of the workflow, manufacturing cloud receives the simulation results and sends them to users. Then the devices and VMs (computing resources) are released accordingly.

10.4 Description and Formulation of OACR

According to the simplified manufacturing process shown in Sect. 10.3, to build a practical model of optimal allocation of computing resources, the core allocation structure and its characteristics should be emphasized firstly. The structure of OACR gives the detailed allocation process of VM management and the

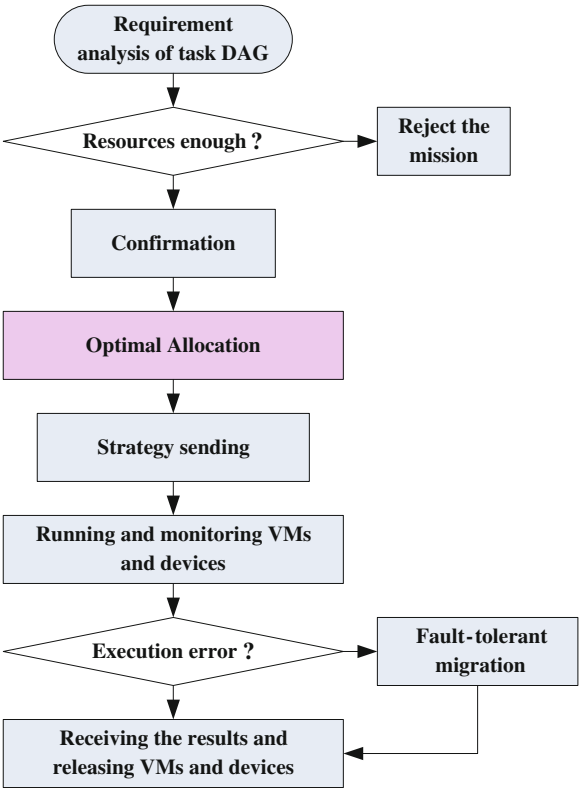


Fig. 10.3 The specific workflow of task execution

distribution characteristic of CRs. Based on that, the communication and topology characteristics of CRs in CMfg are elaborated for further study of the model of OACR.

10.4.1 The Structure of OACR

The OACR of CMfg is composed of three levels: manufacturing task level, virtual resource level and computing resource level, as shown in Fig. 10.4.

On manufacturing task level, assume the tasks of a given MTs set are meta-tasks. *Meta-task* means that the task is inseparable for executing in VMs/CRs, as discussed in Sect. 10.3. For instance, in a multidisciplinary collaborative simulation, each module runs on one VM with user's control and interaction. Each VM is inseparably running on only one CR. So MTs and VMs have the one-to-one mapping relationship.

When MTs' demands are abstracted as virtual resources' demands, the virtual machine manager receive the demand information and allocate available VMs for

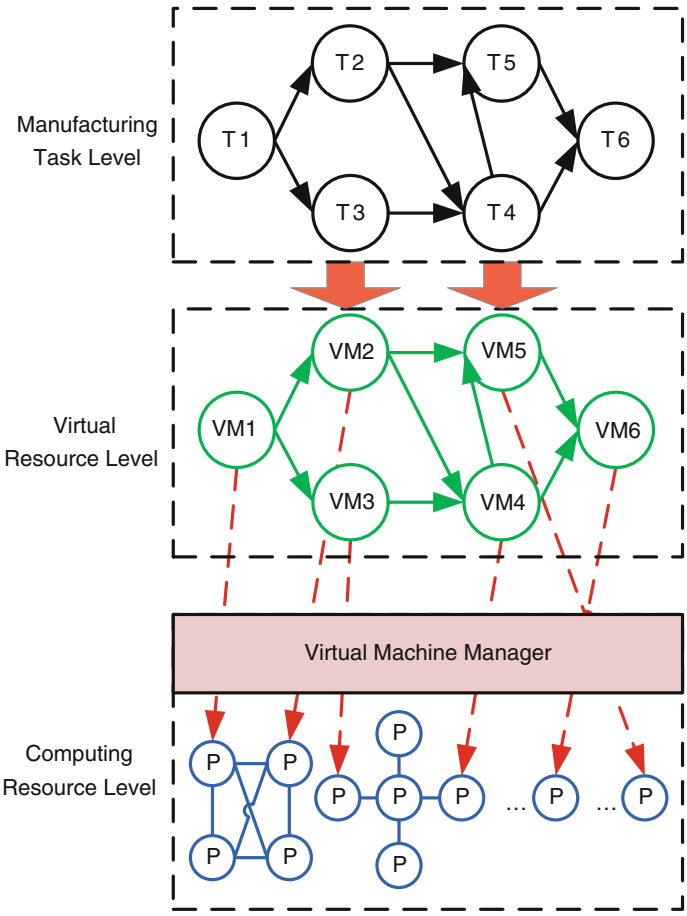


Fig. 10.4 The process framework of OACR

physical manufacturing resources. The physical manufacturing resources can be not only manufacturing/simulation equipments, but also computing resources. Each of manufacturing/simulation equipment needs a CR to control and monitor. Thus, all VMs are supported by CRs. They form the virtual resource level and support the running of MTs. Because the customized MTs are applied by user, the constraints of VMs (e.g., the demand of memory size, computing speed, communication link and bandwidth, etc.) could be obtained at the same time.

As shown in Fig. 10.4, the mapping of manufacturing task level and virtual resource level is the foundation of OACR, and the mapping of virtual resource level and physical resource level is central to the optimization. In this chapter, the manufacturing task level and virtual resource level are merged, and the optimal allocation of MTs (or VMs) and CRs under the concrete computation and communication constraints are emphasized.

10.4.2 The Characteristics of CRs in CMfg

In actual operation of VM management, the topology and communication properties of CRs are very important. These factors determine which CRs are most suitable for MTs and which allocation scheme is the most efficient one, and almost all constraints of OACR come from the characteristics of CRs.

(1) Communication network
The CMfg network is different from other enterprise network or public network and compromised by many distributed manufacturing resource around the world. For the sake of facilitate management and extension, master-slave (manager-service) mode is adopted in the platform. As the shoring of foundation, computing resources can dynamically access the platform via Internet. They are managed and controlled by high stable VMs management system. According to their locations, CRs can be divided into multiple subsets. This topology is similar to the classical tree network. Each subset belongs to different provider who has full authority and obligation to operate and maintain it. The subsets could be mesh/star topology cluster or independent PCs. Due to the different topologies of CRs' subsets, the transmission in group can be half-duplex, full-duplex or busses. With the development of the high speed Ethernet switch, transmission among groups are all full-duplex.
(2) Communication ports
Generally, the port communication of master-slave system can be classified as single-port mode [56] and multi-port mode [57]. *Single-port* mode means that the network central node can only send or receive limited-byte message to/from one slave node in a given period of time. On the contrary, in *multi-port* mode the network central node can send or receive limited-byte message to/from one or more slave nodes in a given period of time. In CMfg, multi-port communication mode is adopted in CRs and the platform.

However, in this multi-port mode, owing to the complex and frequent intercommunication among CRs for a large number of MTs, the amount of transmit data from multi-port to the central node must be huge, which is called *periodic burst* or *data surge*. Periodic burst can cause packet loss and network congestion. In order to avoid this, the general port transmit mechanism of cloud computing is adopted in CMfg, that is, large caches are allocated in the receive direction of switches while small caches are allocated in the send direction to control the flow burst. In this case, the critical cache in the receive direction for preventing data surge and the relationship with the communication time between CRs should be particularly considered.
(3) Communication bandwidth
Associating with multi-resources around the world, the core network protocol of CMfg is still TCP/IP mode (with two-sided communication type [32]. In TCP/IP protocol, Data are usually divided into small packets and transmitted one by one. If one of the packets is not arriving, the packet will be resent or the congestion control strategy will be loaded in the network. Then the data transfer rate will be slower. Though TCP/IP protocol is efficient in short distance transmission, it may

cause delay or packet loss in the large-scaled remote communication in CMfg. Thus, in gigabit network, long-distance communication between CRs will lead to delay at hundred milliseconds scale and small probability packets loss. This makes the actual transfer rate be only about one-tenth of the original bandwidth or even smaller. Therefore, with existing communication technologies, the transfer rates of remote communication among CRs can only reach ten to hundreds Mbps.

10.4.3 The Formulation of the OACR Problem

The above-mentioned structure and main characteristics clearly reflect the high heterogeneity and dynamics of optimal allocation of computing resources. It comes from the traditional models of task scheduling but is more complex than the traditional ones. For describing the model of OACR in formalization, the formal descriptions of tasks and computing resources in traditional task scheduling are shown as follows. And based on the traditional definitions, the new model of OACR is presented then.

(1) Traditional models of tasks and CRs

In general task scheduling problem, the tasks and the multiprocessor system are defined as follows.

Definition 1 The tasks set in multiprocessor system can be presented as a weighted directed acyclic graph (DAG), $G = (V, E, c, w)$. The set $V = \{v_i | i = 1 : v, v = |V|\}$, where v_i represents the task of the set V, and v is the cardinality of nodes. The set $E = V \times V$, $e = |E|$is the number of edges, and $e(ij) \in E$ represents the communication between v_i and v_j. *w(i)* represents the computation cost of v_i. $c(ij) \in c$ represents the communication cost of the directed edge *e(ij)*. If there is no communication between v_i and v_j, then $e(ij) = c(ij) = 0$.

Let the predecessor tasks set of v_i be *pred*(v_i), and the successor tasks set be *succ*(v_i). The node with no predecessor task $pred(v_i) = \emptyset$ is named *source* node, and the node with no successor task $succ(v_i) = \emptyset$ is called *sink* node. They all strictly observe the tasks' priority rules. It means a node can only be started after all its parent (preceding) nodes are finished.

Definition 2 The multiprocessor system, $M = (P, s, bw)$, consists of a finite set of processors $P = \{p_k | k = 1 : p, p = |P|\}$ which are connected by a communication network. The notation $s = \{s(k) | k = 1 : p, p = |P|\}$represents the computing power of processors, $bw = P \times P$ represents the bandwidth between processors, and $bw(kl) \in bw$ is the bandwidth between p_k and p_l. If the system is homogeneous, the processor's computing power and their bandwidths are all equal, that is $\forall k, l \in [1, p], k \neq l \Rightarrow s(k) = s(l),\ bw(k) = bw(l)$. Heterogeneous systems are then contrary.

In these models, processors are usually all directly connected, and the tasks are non-preemptive. If two tasks are carried by the same processor, their communication cost is 0, and it assumed that the transmission rate of computing resources equal to the bandwidth (the ideal value).

(2) The new models of OACR in CMfg

According to the characteristic of CRs, the uncertain topology can be simplified as shown in Fig. 10.5. The above-mentioned CRs subsets are simplified as different groups. Different topologies in groups can be reflected by the communication links among CRs. That is to say, with different topologies, CRs in the same group connected with each other through different communication links by local connection, and CRs in different groups are connected by switches via Internet. Stand-alone PCs can be classified as a special group. They are connected with each other directly via Internet.

In theory, the biggest difference between general computing tasks and MTs are whether they are controlled by and interacted with users during execution time. Control and supervision are generally implemented by multi-thread in CRs. The MTs' computation costs vary according with users' interactions. Because of the frequent control and supervision in MTs, the execution times might be much longer. How long it will be depends on how many interactions and supervisions during MTs' execution. For considering this, the new MTs model is defined as:

Definition 3 The MTs set in CMfg can be presented as a weighted directed acyclic graph (DAG), $G = (V, E, c, w, oper\_p, su\_p)$. The definition of $V = \{v_i | i = 1 : v, v = |V|\}$, $E = V \times V$, w and c are the same as the traditional task model (Definition 1). The set $oper\_p = \{oper\_p_i | i = 1 : v, v = |V|\}$ and $su\_p = \{oper\_p_i | i = 1 : v, v = |V|\}$ represent relative interoperation-to-computing ratio and relative supervision-to-computing ratio separately. That is to say, the estimated cost of interoperation $oper = c \times oper\_p$ and the estimated cost of supervision $su = c \times su\_p$.

Then the total cost of each node in G can be calculated as $W(i) = w(i) \times (1 + oper\_p(i) + su\_p(i))$. If there's no interaction or supervision in v_i, then $oper_p(i) = 0$or $su\_p(i) = 0$. These two factors can clearly reflect the users demands for interaction in MTs and the new model can then be more practical.

With the users' interaction and large-scaled computation and communication costs, the installments and involvements of VMs can be ignored from both communication and computation perspective. On the basis of the topology and the characteristics of CRs, the new CRs model can be defined as follow.

Definition 4 The CRs system model of CMfg is given by $M = (P, s, rou, bw, mem, buf, rel)$, where

- $P = \{p_{kl} | k = 0 : m, l = 1 : n_k\}$ represents the CRs set, in which m is the number of CRs groups, and n_k is the resources number in group k. Let $k = 0$ represent the stand-alone PCs, and n_0 represent the number of these stand-alone PCs. Therefore, the total quantity of CRs is $|P| = \sum_{k=0}^{m} n_k$.
- $s = \{s(kl) | k = 0 : m, l = 1 : n_k\}$ represents the computing power (the computing speed) of the CRs set.
- $mem = \{mem(kl) | k = 0 : m, l = 1 : n_k\}$ represents the available memory volume of CRs, in which*mem(kl)* varies dynamically with the task running. Its memory volume is reduced accordingly, when a task (VM) is assigned to the CR.

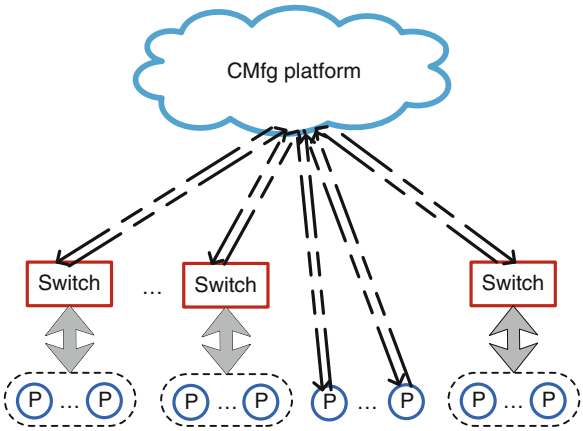


Fig. 10.5 The simplified topology of CRs

- $bw = \{bw(kl)|k = 1 : m, l = 1 : n_k\}$ represents the bandwidth between CRs and switch in each group. Considered the simplified topology (Fig. 10.2), the access bandwidths of switches to Internet are defined as $BW = \{BW_i|i = 1 : m\}$. Due to the stand-alone PCs are connected via Internet directly, let $bw_0 = \{bw_0(kl)|k = 1, l = 1 : n_0\}$ be their bandwidth to Internet. Owing to the bandwidths in groups are generally gigabit, $\forall k \in [1, m], l \in [1, n_k] \Rightarrow BW_k \ll bw(kl)$.
- $rou = \{rou_i(kl)|i = 1 : m, k \neq l, k, l \in [1, n_i]\}$ represents the communication route between p_{ik} and p_{il} in local connection. Because the subsets of CRs are dynamic and complex, the route and bandwidths of the communication between two CRs needs to be calculated by a specific way when the subset is accessed. So the concrete topologies in groups are not considered in this model. It is assumed that the communication routes and bandwidths among CRs are previously figured out by some kinds of routing algorithms. The simplified communication route $rou_i(kl) = \{link_1, \cdots, link_r\}$[32] varies with different topologies, and the bandwidth of $rou_i(kl)$ is defined as $bw(rou_i(kl)) = \min\{bw(link_1), \cdots, bw(link_r)\}$.
- $buf = \{buf(kl)|k = 1 : m, l = 1 : n_k\}$ represents the buffer size of the switch communication ports in each group. According to Davidovi et al. [30], it assumed that the highest tolerable abrupt data of each port to be:

$$D(kl) = buf(kl) + buf(kl)\frac{BW_k}{bw(kl) - BW_k} = buf(kl)\frac{bw(kl)}{bw(kl) - BW_k} \tag{10.1}$$

- $rel = \{rel(kl) = (rel\_p(kl), rep\_t(kl))|k = 0 : m, l = 1 : n_k\}$ represents the reliability of the CRs set. The reliability of CR means the probability that the computing resource fails to connected in the consequence of the communication link or occurrence of another MTs set which leads to pause computation for some times. So *rel_p(kl)* represents the probability and *rep_t(kl)* represents

the predicted failure duration time of CRs. Then $\forall k \in [0,m], l \in [1,n_k] \Rightarrow rel\_p(kl) \in [0,1]$.

In this model, *rou* and *bw* are used to represent the local connections and the remote connections separately. Therefore, the OACR model can be described as $S = (G,M)$, where $G = (V,E,c,w,oper\_p,su\_p)$ represents the MTs and $M = (P,s,rou,bw,mem,buf,rel)$ represents the CRs.

(3) The constraints and objective function of OACR

Based on the structure described before, four issues of CRs are considered in this chapter.

(1) The minimum acceptable memory size $MEM_{\min}(i)$ for task v_i;
(2) The minimum acceptable reliability $REL_{\min}(i)$ for task v_i;
(3) The minimum acceptable computing speed $EXE\_SPEED_{\min}(i)$ for task v_i;
(4) The longest acceptable communication time $COM\_TIME_{\max}(ij)$ for task v_i, usually it is much looser than the above three constraints.

When a MTs set $G = (V,E,c,w,oper\_p,su\_p)$ is applied to CMfg platform, the system $M = (P,s,rou,bw,mem,buf,rel)$ will provide right CRs for it. Let *k(i)* and *l(i)* be the group number and the position of the selected CR for task v_i, and let *p_load(k(i)l(i))* be the load of the selected CR for task v_i, which is measured by MTs per CR. Then the constraints of each selected CR can be described as:

- When multi MTs $\{v_1 \cdots v_n, n < v = |V|\}$ select the same CR P_s, if $mem(s) \geq \sum_{i=1}^{n} MEM_{\min}(i)$ then $p\_load(s) = 1$, else, MTs are needed to queue for execution, that is, $p\_load(s) = n$;
- $\forall i \in [1,v]$, the computation speed of the selected CR for task v_i satisfied: $s(k(i)l(i))/p\_load(k(i)l(i)) \geq EXE\_SPEED_{\min}(i)$, and then the execution time of task v_i can be expressed as $EXE\_TIME(i) = W(i) \times p\_load(k(i)l(i))/s(k(i)l(i))$;
- $\forall i \in [1,v]$, the reliability of the selected CR for task v_i satisfied: $rel\_p(k(i)l(i)) \geq REL_{\min}(i)$,

The constraints of the communication ability of the selected CRs can be represented as $COM\_TIME(ij) \leq COM\_TIME_{\max}(ij)$. It can be divided into two cases. $\forall i,j \in [1,v], i < j$ (v_i is the predecessor task of v_j), let $COM\_TIME\_s(ij)$ be the data sending time between the two tasks, and $COM\_TIME\_r(ij)$ be the data receiving time between two tasks.

Case 1 when v_i and v_j are in the same CRs group, $k(i) = k(j) = k \neq 0$. Without considering the reliability factors, the sending time of v_i is equal to the receiving time of v_j, that is:

$$COM\_TIME\_s(ij) = COM\_TIME\_r(ij) = COM\_TIME(ij) = c(ij)/rou_k(l(i)l(j)) \tag{10.2}$$

With the addition of the *rel* factor, let *t(x)* be the average communication time between v_i and v_j which originally needs x seconds of processing. If the CR $p_{k(j)l(j)}$ doesn't fail halfway, then the communication time needs *1 + t(x-1)* seconds, but if it fails at midway (with the probability *rel_p(k(j)l(j))*), then it needs to wait *rep_t(k(j))(l(j))* seconds and also need another *t(x)* seconds to complete the communication. Therefore it has:

$$t(x) = (1 - rel\_p(k(j)l(j)) * (1 + t(x-1)) + rel\_p(k(j)l(j)) * (t(x) + rep\_t(k(j)l(j))) \tag{10.3}$$

$$t(x) = 1 + t(x-1) + \frac{rel\_p(k(j)l(j)) * rep\_t(k(j)l(j))}{1 - rel\_p(k(j)l(j))} \tag{10.4}$$

Since $t(0) = 0$, it can be written as:

$$t(x) = x(1 + \frac{rel\_p(k(j)l(j)) * rep\_t(k(j)l(j))}{1 - rel\_p(k(j)l(j))}) \tag{10.5}$$

According to Eq. 10.5, the communication time between v_i and v_j can be expressed as:

$$COM\_TIME(ij) = (1 + \frac{rel\_p(k(j)l(j)) * rep\_t(k(j)l(j))}{1 - rel\_p(k(j)l(j))}) * \frac{c(ij)}{rou_k(l(i)l(j))} \tag{10.6}$$

Case 2 when v_i and v_j are in different CRs groups,

- If $c(ij) < D(k(i)l(i))$, without considering the reliability, the sending time of task v_i is equal to:

$$COM\_TIME\_s(ij) = c(ij)/bw(k(i)l(i)) \tag{10.7}$$

and the receiving time of task v_i is equal to:

$$COM\_TIME\_r(ij) = \left\{ \begin{array}{l} \frac{c(ij)}{\min\{BW_{k(i)}, BW_{k(j)}\}}, k(i) \neq k(j) \neq 0 \\ \frac{c(ij)}{\min\{BW_0(l(i)), BW_{k(j)}\}}, k(i) = 0 \\ \frac{c(ij)}{\min\{BW_{k(i)}, BW_0(l(j))\}}, k(j) = 0 \\ \frac{c(ij)}{\min\{BW_0(l(i)), BW_0(l(j))\}}, k(i) = k(j) = 0 \end{array} \right\} = COM\_TIME(ij) \tag{10.8.}$$

After adding the reliability factors, according to Eq. 10.5, the sending time of task is unchanged, but the receiving time is changed as:

$$\begin{aligned} COM\_TIME\_r(ij) &= (1+\frac{rel\_p(k(j)l(j))*rep\_t(k(j)l(j))}{1-rel\_p(k(j)l(j))}) \\ &* \left\{ \begin{array}{l} \frac{c(ij)}{\min\{BW_{k(i)},BW_{k(j)}\}}, k(i)\neq k(j)\neq 0 \\ \frac{c(ij)}{\min\{BW_0(l(i)),BW_{k(j)}\}}, k(i)=0 \\ \frac{c(ij)}{\min\{BW_{k(i)},BW_0(l(j))\}}, k(j)=0 \\ \frac{c(ij)}{\min\{BW_0(l(i)),BW_0(l(j))\}}, k(i)=k(j)=0 \end{array} \right\} \\ &= COM\_TIME(ij) \end{aligned} \tag{10.9}$$

- If $c(ij) > D(k(i)l(i))$, the sending rate of task v_i must be reduced. According to Eq. 10.1, the sending rate $ssend(i) = c(ij) * BW_{k(i)}/(c(ij) - buf(k(i)l(i)))$. So the sending time of v_i is changed as $COM\_TIME\_s(ij) = (c(ij) - buf(k(i)l(i)))/BW_{k(i)}$. Yet the receiving time of v_j would remain as Eq. 10.9, and $COM\_TIME\_r(ij) < COM\_TIME\_s(ij)$.

Based on these constraints, let the start time of task v_i be *START_TIME(i)*, the execution time of v_i be *EXE_TIME(i)*, and the finish time of v_i be *FINISH_TIME(i)*, then:

$$START\_TIME(j) = \max_{i\in pred(v_j)}\{COM\_TIME\_r(ij)\} \tag{10.10}$$

$$\begin{aligned} FINISH\_TIME(j) &= START\_TIME(j) + EXE\_TIME(j) \\ &+ \max_{i\in succ(v_j)}\{COM\_TIME\_s(ij)\} \end{aligned} \tag{10.11}$$

Therefore the temporal relation between two adjacent tasks is as shown in Fig. 10.6. Note that the source node v_1 do not need to receive data, so $START\_TIME(1) = 0$, and the sink node's sending time is also the MTs submission time, that is:

$$\begin{aligned} COM\_TIME(v) &= TASK\_SUBMISSION\_TIME(V) \\ &= \left\{ \begin{array}{ll} \frac{c(v)-buf(k(v)l(v))}{BW_{k(v)}} & , if\ c(v) > D(k(v)l(v)) \\ \frac{c(v)}{BW_{k(v)}} & , if\ c(v) < D(k(v)l(v)) \end{array} \right. \end{aligned} \tag{10.12}$$

where *c(v)* represents the submission data of MTs.

In conclusion, the execution time of the whole MTs set is:

$$TOTAL\_TIME(V) = FINISH\_TIME(v) - START\_TIME(1) = FINISH\_TIME(v) \tag{10.13}$$

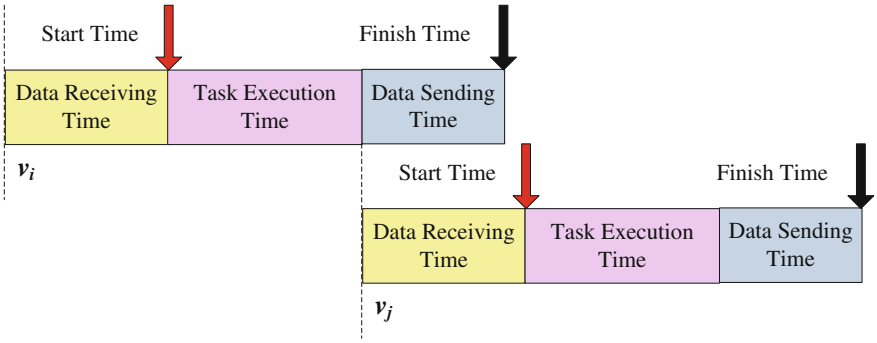


Fig. 10.6 The temporal relation between two MTs

As the constraint of memory size of CRs is embodied in the constraint of computing speed and the reliability factors is embodied in the constraints of communication in CRs, the optimal object function and the constraints of OACR $S = (G, M)$ can be summed up as:

$$\begin{cases} MINIMIZE\ TOTAL\_TIME(V)\ SUBJECT\ TO \\ \forall i \in [1, v],\ \frac{s(k(i)l(i))}{p\_load(k(i)l(i))} \geq EXE\_SPEED_{\min}(i) \\ \forall i, j \in [1, v],\ COM\_TIME(ij) \leq COM\_TIME_{\max}(ij) \end{cases} \tag{10.14}$$

(4) Problem complexity

Traditional task scheduling problems are proved to be NP-complete problems. To prove the complexity of OACR, two definitions are introduced in this section according to Gawiejnowics [58].

Definition 5 [58] (A polynomial-time transformation): A polynomial-time transformation of a decision problem P' into a decision problem P ($P' \propto P$) is a function $f : D_{P'} \rightarrow D_P$ satisfying the following two conditions:

(a) the function can be computed in polynomial time;
(b) for all instances $I \in D_{P'}$, there exists a solution to I if and only if there exists a solution to $f(I) \in D_P$.

Definition 6 [58] (An NP-complete problem): A decision problem P is said to be NP-complete, if $P \in NP$ and $P' \propto P$ for any $P' \in NP$.

Theorem 1 the OACR problem is NP-complete problem.

Proof In the OACR problem, the task quantity of a MTs set $v = |V|$is less than the processors number of a CRs set $p = |P|$. One processor can carry multi tasks.

(1) When $1 < N_p < v$, choosing N_p suitable processors from p resources has $C_p^{N_p}$ solutions. After choosing these N_p processors, the mapping of v meta-tasks and N_p resources turn into the traditional scheduling problem. In this situation the OACR problem can be reduced to the traditional scheduling problem. According to definition 6, the traditional scheduling problem is NP-complete, so the OACR problem is NP-complete.

(2) when $N_p = v$, choosing v suitable processors from p resources has C_p^v solutions. Afterwards, the mapping between v meta-tasks and v computing resources can be converted to TSP (Traveling Salesman Problem), which is the full permutation problem. In this situation, the OACR problem is reduced to TSP problem. From Definition 6, TSP problem is NP-complete, thus the OACR problem is NP-complete, too.

The above discussions contain all cases of the OACR problem, so Theorem 1 is true. Q.E D.

From the point of the solutions, let n be the total amount of CRs in the CMfg platform. and let v be the task number of an applied MTs set. For the case of no time constraints, each task has n choices. So the size of the solution space is n^v. If some one want to find the best solution one by one in the entire solution space, then they need $O(n^v)$ steps to complete. It is a huge calculation. For example, if there're 100 CRs and 5 MTs, the solution space is 5^{100}. It's a very huge number for calculation. At present, no deterministic algorithms can solve it in polynomial time. So, intelligent algorithms are introduced in this chapter.

10.5 NIA for Addressing OACR

The most frequently used intelligent algorithms for the traditional task scheduling are genetic algorithm (GA) [43, 44] and ant colony optimization algorithm (ACO) [47, 48]. Besides, immune algorithm (IA) has shown great potential in combinatorial optimization problems [59, 60]. They are widely used in various kinds of scheduling problems and their basic processes are shown in Fig. 10.7. Based on these three classical intelligent algorithms and with the consideration of the complex of OACR, a new improved niche dynamic IA (NDIA) is proposed in this chapter for better solutions. These four algorithms will then be used and generally analyzed for addressing the OACR problem in detail.

10.5.1 Review of GA, ACO and IA

GA is an adaptive global optimization stochastic search algorithm which is inspired by the principle of evolution and natural genetics. With a set of structured populations which represented the candidate solutions of the problems, it combines the survival of the fittest among populations (selection), the structured yet randomized information exchange (crossover) and the random bit mutation.

Firstly, roulette wheel selection is used commonly in the standard GA. It is performed by randomly picking a certain amount of populations according to their fitness values to form a new group of populations. The one with higher fitness value occupied higher probability of selected. Secondly, each two of the selected

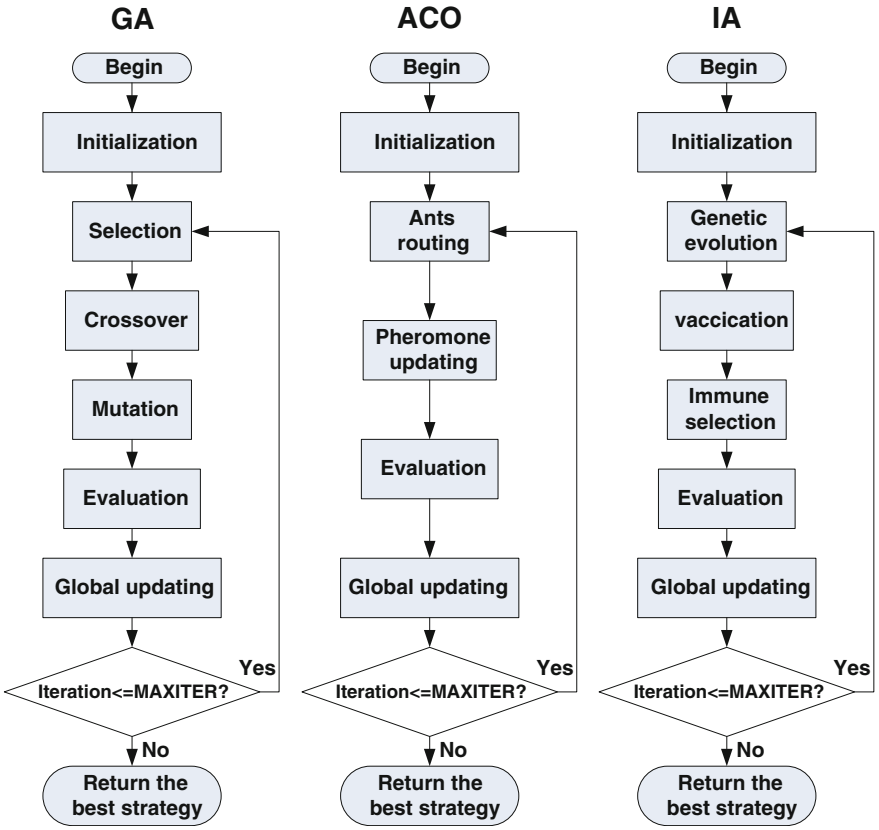


Fig. 10.7 The process of GA, ACO and IA

populations exchange parts of their gene-bits in the crossover operation. So two new genetic chromosomes are generated and the better gene bits go into the next generation. Thirdly, the mutation operation randomly changes some gene bits of populations with a certain probability for increasing the diversity. After the above three steps, if the new best population is better than the old one, then it would evaluated by the new one, or the best population record will remain unchanged. This process will repeat and then terminate when the maximize generations are reached or the optimal solution is found.

ACO is a kind of swarm intelligence algorithms which takes inspiration from the social behaviors of ant colony. It combines ants' routing and pheromone update. The original intention of ACO is to solve the complicated path optimization problems, such as TSP, and edge scheduling.

In the process of routing and finding foods, ants deposit pheromone on the path they have walked in order to mark some favorable path and broadcast the information. The longer the path, the lower the density of pheromone is. Other ants can

Table 10.1 The characteristics of GA, ACO and IA

Algorithm	Year	Mechanism	Priori knowledge	Global convergence	Control parameters
GA	1975	Biological evolution	Needless	Weak	Less
ACO	1992	Ants behavior	Need	Strong	More
IA	2000	Immune system	Need	Strong	Medium

perceive this pheromone and recognize its density. They have a large probability to select the path which has the greater pheromone density. Then a kind of information positive feedback is formed. The pheromone density on the optimal path will become higher, while the pheromone density on other paths will reduce as the time goes by. Finally the whole colony will find the optimal path.

With this inspiration, the priori knowledge is introduced and formed the standard ACO. That is to say, ants are finding path not only in the light of the pheromone, but also according to the priori rules (knowledge) of the problems. As the same with GA, the whole process continues many evolution times until the ants find the optimal path (solution) or the number of evaluation steps reach a predefined value.

IA is a kind of evolutionary programming which based on the immune system in biotic science. With the introduction of the concepts and the characteristics of antigen recognition, immunological memory and immune regulation, diversified immune algorithms are presented. The immune algorithm proposed by Lei Wang [59, 60] is a typical and efficient one. In this chapter, it will be applied in OACR and IA here just indicates the algorithm in [59, 60].

More specifically, IA is a convergence of immune theory and genetic algorithm. It contains genetic evolution, immune vaccination and immune selection, but first of all, antigen extract and vaccine selection according to the feature information of problem is the most important part of this algorithm. It is a core rule to lead the population evolution in the right direction. Then, the population initialization and the genetic evolution are all the same as the standard genetic algorithm. After selection, crossover and mutation, new populations are vaccinated by antibodies. That is injecting priori knowledge into the new populations in some degree for improving their fitness values. (The priori knowledge in IA is the same as in ACO.) Then, new populations are selected by immune selection operation according to the choosing rules of simulated annealing. Three steps will repeat until the ending conditions are meeting.

All of the above-mentioned intelligent algorithms are evolved with a number of cycles by their own mechanism. As shown in Table 10.1, only GA does not need the priori knowledge with less control parameters. Without other improvement strategies, its global convergence is weak, but its robustness is quite good. ACO and IA are both need the direction of priori knowledge with good global convergence. Yet the control parameters of ACO are more than IA's.

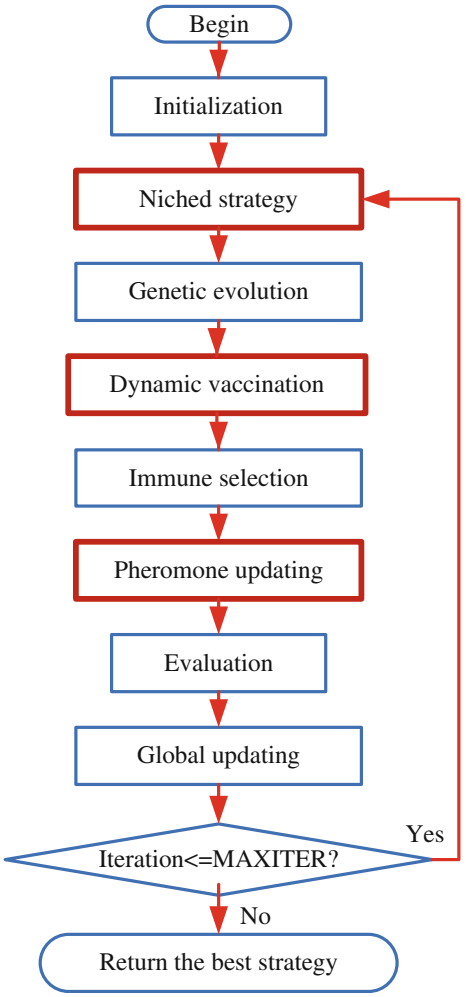


Fig. 10.8 The framework of NDIA for addressing OACR

10.5.2 The Configuration OfNIA for the OACR Problem

Inspired by the above three algorithms, the improved niche IA takes the techniques of pheromone guide from ACO and the ecological niche strategy. Its framework is shown in Fig. 10.8.

Compared with IA (as shown in Fig. 10.7), the niche strategy, dynamic vaccination and pheromone updating strategy are added in NDIA. Niche strategy is used for improving exploration during searching, dynamic vaccination and pheromone updating strategy is taked for further improving the exploitation and searching direction with the dynamical consideration of both computation and communication in OACR. The genetic evolution just adopts the standard roulette wheel strategy, single-point crossover and mutation. The improvement of initialization,

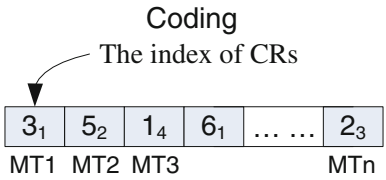


Fig. 10.9 The real number coding for OACR

the object function and new improved strategies in NDIA for solving OACR are elaborated as follows.

(1) Initialization

For solving OACR problem, real number coding is used in the experiments. Real number coding can avoid the encoding/decoding process, improve the accuracy and reduce the complexity of the algorithm. As shown in Fig. 10.9, the sequence number of gene bits is denoted as the serial number of MTs, the numbers in the gene bits represent the index of CRs and their subscripts represent the group indexes of CRs. In other word, each gene bit occupies two integer bits. This kind of coding method takes less space and is more intuitive and simple.

(2) Object function in Evolution

Because the standard GA with the roulette wheel strategy is commonly used to find the individual with the maximize fitness value. The fitness evaluation function of OACR in all intelligent algorithms is set as:

$$\max f = \frac{Const}{TOTAL\_TIME(V)} \tag{10.15}$$

Const is a constant which makes the object fitness value in the algorithms neither too large nor too small.

(3) Niche strategy

For improve the diversity and balance the exploration and exploitation of the algorithms, the technology of ecological niche is introduced in it. In NIA, the hamming distances D_{ij} between two individuals should be calculated before the implementation of genetic evolution, as shown in Eq. 10.16.

$$D_{ij} = \left\| X_i - X_j \right\| = \sqrt{\sum_{k=1}^{N} (x_{k,i} - x_{k,j})^2} \tag{10.16}$$

where X_i and X_j represent individual i and individual j, and $x_{k,i}$ and $x_{k,j}$ represent the gene bits of each individuals separately. If D_{ij} (between individual i and j) is less than a pre-set parameter L, then the individual with lower fitness value will multiple a penalty function to make it more lower. This action could wipe off the similar individuals and protect the diversity of the population to improve the search ability.

Because of the definition of the maximum object function, here the penalty function is set as $f = 10^{-5}$ and let the parameter L to be v which is the size of the individual in algorithms (MTs' number).

(4) Dynamic vaccination and pheromone updating

As is known to all, antigen extraction and vaccine selection is the core factor of IA and it usually comes only from the priori knowledge (i.e. heuristic information) of the problems. If the priori knowledge is extracted inappropriately, the algorithm will evolve in the wrong direction and no feasible solution can be found. However, the priori knowledge in the complex problem is usually complex and varying with different situations. For example, both the computation and communication (node and edge factors) should be considered in OACR especially when the computation rate of MTs is equal to its communication rate. The incidence relations among tasks are very complex and the computation and communication power of CRs are varying dynamically. The extracting and the rate selection of the two factors are therefore hard. This directly influences the efficiency of IA in solving the global optimal solutions.

To avoid this problem, and considering the dynamic change of the memory size, communication bandwidth and reliability constraints of CRs, we present the new dynamic vaccination strategy. That is, extraction and calculating the heuristic information (η_{ij}) of allocating the CR p_j to the MT v_i needs real time in each evolutionary cycle. It is time consuming but can obtain higher accuracy result in the scheme, and for simplification, the heuristic information function is set as:

$$\eta_{ij} = \frac{\left(\frac{s(k(i)l(i))}{p\_load(k(i)l(i))}\right)^{\alpha} * (bw(\mathrm{k(i)}l(i)))^{\beta}}{\left(1 + \frac{rel\_p(k(j)l(j) * rep\_t(k(j)l(j)}{1 - rel\_p(k(j)l(j)}\right)^{\gamma}} \tag{10.17}$$

where α, β and γ represent the importance of the execution speed, communication bandwidth and reliability of CR in the heuristic information and they satisfied $\alpha, \beta, \lambda \in [0, 1]$. In experiments, the value of these parameters will be tested and discussed for better solution.

Besides, for further improving the searching direction, the pheromone of ACO is brought in NDIA in this chapter. That is to say, the improved NDIA extracts antigen and vaccine not only by the priori knowledge, but also by the pheromone which is released by the previous best individual of the whole populations. As a supplement, the pheromone increased the experiential guidance and made a positive feedback in IA. To put it more specifically, let τ_{ij} be the density of pheromone of mapping MT v_i to CR p_j and η_{ij} be the priori knowledge (i.e. the heuristic factor) of the problem. The vaccine can then be expressed as:

$$vaccine = (\tau_{ij})^{\varphi} (\eta_{ij})^{\phi} \tag{10.18}$$

where τ_{ij} is updating as the same as in ACO, and $\varphi, \phi \in (0, 1)$ represent the strength factors of τ_{ij} and η_{ij} separately. If φ is too larger than ϕ, then vaccine will

be directed by the experience of populations and result in low searching ability and low convergence. But on the contrary, too larger ϕ will also changed vaccine to static and lead to premature. So, according to ACO, the rates of pheromone and heuristic information in Eq.10.17 in this chapter are set the same as ACO (i.e. $\varphi = 1$, $\phi = 5$).

Then at the vaccination step, the one or more gene-bits of the selected populations will be changed as the one with the highest *vaccine* at a certain rate. This strategy can improve the convergence, increase the robustness and simplify the previous vaccine extraction and calculating in algorithms.

10.5.3 The Time Complexity of the Proposed Algorithms

The time complexity of the intelligent algorithms is dynamically varied with different problems. Let n be the scale of the population, m be the scale of CRs in CMfg platform and v be the scale of the MTs set applied by user. The algorithms' complexities in each cycle (or generation) are shown in Table 10.2.

GA does not need the heuristic information to direct its evolution. In selection, the complexity of the roulette wheel strategy in the best situation is O(n), and its worst complexity is O(n^2). Due to the worst case complexity of the algorithm is the upper bound of run time. The complexities in Table 10.2 just mean the worst case complexities.

In ACO, because ants' routing needs to calculate the priori knowledge in each cycle, finding a suitable CR for each task then needs m step to get all of the heuristic information of CRs. With n populations and v tasks, its complexity is O(nmv).

The same as the ACO, IA needs to find a certain number of population and vaccinates them. In vaccination, the load and memory of each CR should be calculated according to the mapping of MTs, so the complexity of this operator is O($n(m + v)$). Then the immune selection will decide if the new populations can be kept in the next generation according to the choosing rules of simulated annealing. For n new populations (at most) and v tasks, the complexity is O(nv).

Based on IA, the complexities of additional strategies in NDIA are also shown in Table 10.2. Owing to the calculation of hamming distance among populations, the niche strategy's complexity is O(n^2). The dynamic vaccination in each evolutionary cycle is O(m), and the pheromone updating strategy is O(mv). So in theory, the additional operators in NDIA did not increase the complexity of the algorithm.

The complexities of above-mentioned four algorithms when $n \to \infty$ and $m \to \infty$ and $v \to \infty$ are also proposed as Table 10.2 shows. If the population size of the algorithms is large, the complexity of ACO (O(n)) is the lowest. When the scale of CRs $m \to \infty$, then the lowest complexity is O [1] in GA. However, when the scale of MTs $v \to \infty$, then the complexities of the four algorithms are all the same (i.e. O(v)).

Table 10.2 The time complexities of the three algorithms

Algorithms	The time complexities of operators			$n \to \infty$	$m \to \infty$	$v \to \infty$
GA	Selection	Crossover	Mutation	$O(n^2)$	O(1)	$O(v)$
	$O(n^2)$	$O(n)$	$O(nv)$			
ACO	Ants' routing	Pheromone updating		$O(n)$	$O(m)$	$O(v)$
	$O(nmv)$	$O(mv)$				
IA	Genetic evolution	Vaccination	Immune selection	$O(n^2)$	$O(m)$	$O(v)$
	(pending)	$O(n(m + v))$	$O(nv)$			
NDIA	IA evolution	Dynamic vaccination and Pheromone updating	Niched strategy	$O(n^2)$	$O(m)$	$O(v)$
	(pending)	$O(mv)$	$O(n^2)$			

10.6 Configuration and Parallelization of NIA

As mentioned in Chap. 5, there are many topologies for the parallelization of intelligent optimization algorithm. No matter with large-scaled or small-scaled computing resources, communication on one hand is a critical issue for preserving the overall performance of parallel algorithm, on the other hand is also a decisive factor the total time consumption of searching process. Therefore, the control of individual exchange is the most important thing in the parallelization configuration.

As is known, during the evolutionary process, if the sub-populations in different nodes are consistent and distributed in a small solution space, the communication is needless. Too frequent exchange in this situation is very likely to get premature convergence. On the contrary, if the sub-populations in different nodes are dynamic and distributed in a large solution space, then the communication is required. In this case, foreign excellent individuals will always bring good information and push the whole population evolving in a good direction. Otherwise, sub-population will do evolution on their own with a lot of repetitions and no convergence at all. Inspired by the use of prior knowledge in many adaptation strategies, we present a new adaptive ways for control the communication step and make exchange only when the whole population is high diversity.

Firstly we need to select a suitable connected topology.

1. Connected topology

The common used topologies include ring, grid and full-mesh and so on. Normally, with fixed number of individuals in the whole population, dense connection topologies such as full-mesh and grid can obtain higher collaborationamong subgroups but with higher communication overhead,vice versa. However, based on general parallel tools - MPI (Message Passing Interface), we found that-MPI_Allgather (the full-connected interaction way) is more efficient than the use of MPI_Send/MPI_Recv or MPI_Isend/MPI_Irecvto implement full communications during$n$ nodes due to the inside optimization by the tools itself. In this situation, full-mesh is quite advantageous.

Thus, full-mesh module is selected to generate the new PNIA. Nonetheless, full-mesh cannot make sure low time consumption during iterations. Considering the time cost of MPI_Allgather, the design of efficient migration mechanism is imperative.

2. New adaptive strategy

In each period, whether to exchange individuals is decided by the overall evolutionary state of the sub-populations. Specifically, migration is not always needed in iterations. If the sub-populations execute many times of iterations but with low evolution and diversity, then migration is needed to introduce outside excellent genes to improve the quality and diversity of sub-populations. Otherwise, migration is needless at all because good evolutionary direction of sub-populations

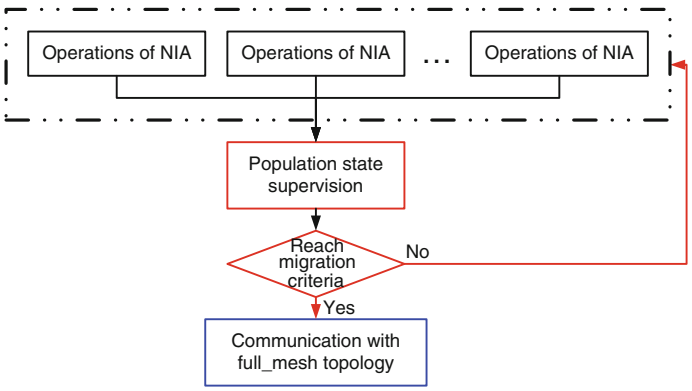


Fig. 10.10 The configuration of population supervision module with full_mesh topology

could easily be disturbed. Not only is the redundant migration wasting time, but also it does not bring good benefits.

Thus, from the whole population point of view, we set one computing node to take in charge of supervising and gathering the state parameter of all sub-populations, as well as master-slave mode, and broadcasting the overall state to all others with Map/Reduce operations in MPI. Whether to do interaction depends directly on the overall state parameter.

As we know, one of the key factors to reflect the evolutionary state is the total number of progression-free generations. It implies that, since the last generation of the improvement of historical best solutions, the generations have been executed without the change of the historical best record. Let the progression-free generation of each sub-population i be G_inva_i. If better solution has been searched in the current generation, then $G\_inva_i = 0$. Otherwise in each generation, execute $G\_inva_i = G\_inva_i + 1$. To avoid extra large value of G_inva_i, we could set $G\_inva_i \in [0, G_{\max}]$ in which G_{max} represents the upper bound of G_inva_i.

According to such variable, the evolutionary state of each sub-population can be easily obtained. With fixed topology modules in configuration, here we only need the overall evolutionary state to control the exchange process. For balance among different sub-population, set the total progression-free generations G_inva_total to be calculated as follows.

$$G\_inva\_total = \sum_{i=0}^{n} G\_inva_i \tag{10.19}$$

where n is the total number of sub-groups. If the total progression-free generation parameter becomes large,then carry out the migration with full mesh topology and specific migration mechanism. Otherwise, stop the migration and keep self-evolution going.

Moreover, set the migration condition (migration frequency) to be MT, then the migration with full-mesh topology is allowed with the probability E_c.

$$E_c = \exp(-\frac{G\_inva\_total}{n \times G_{\max}}) \tag{10.20}$$

It can be seen that if G_inva_total is smaller, $E_c$will be increased, the time of migrations will be decreased, vice versa.

From the perspective of parallelization, the specific pseudo-code of the adaptive stragey with full mesh topology can be represented as follows.

```
For (each sub-population i in parallelization)
    Initialize subpopulation
    generation = 0
    While(generation<= MAX_generation or convergence criterion satisfied)
            generation ++
            Apply algorithm's operators
            Evaluate solutions in the subpopulation
            If (generation % MT == 0)
                    If (i == supervision_node)
                            Reduce G_inva_total = sum_i G_inva_i
                            Broadcast G_inva_total
                    End if
                    If (rand()>exp(-1*G_inva_total / (n*G_max)))
                            Migration with full_mesh_topology
                    End if
            End if
    End while
```

From this, the total parallel efficiency can be improved under the mode of MPI_Allgather (full-connection topology) by means of supervision and adaptive communication.The module of population supervision can be shown in Fig. 10.10.

10.7 Experiments and Discussions

For testing the OACR models, the DAG in Fig. 10.4 and other two kinds of DAGs (the e-Economic DAG and the e-Protein DAG) introduced from [48] are selected, as shown in Fig. 10.11.

Based on the above mentioned four algorithms and the new OACR models, the *CCR* [31] is introduced as the testing factor in this chapter. It is defined as the communication to computation ratio.

$$CCR = \frac{\sum_{e \in E} c(e)}{\sum_{n \in V} w(n)} \tag{10.21}$$

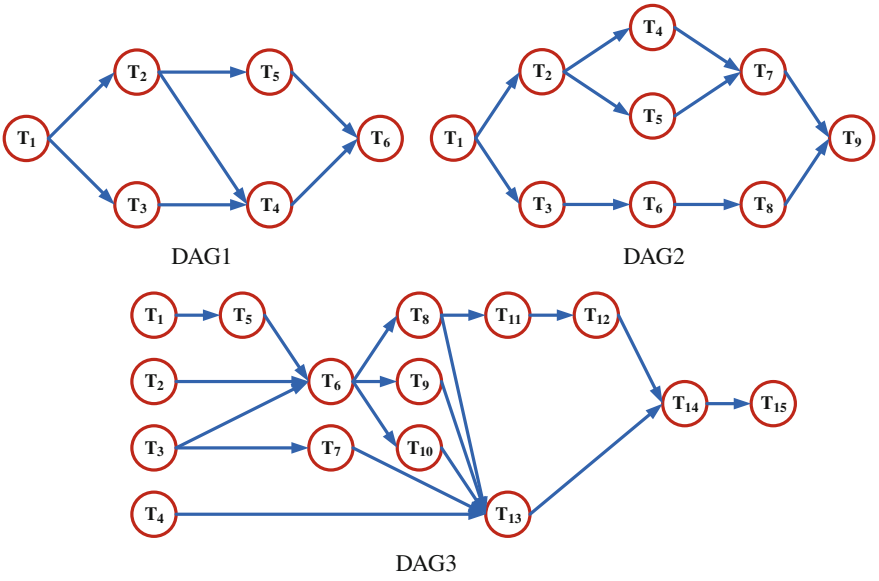


Fig. 10.11 DAGs of the selected MTs

In the experiments, all of the communication costs and the computation costs are randomly generated. Due to the looser communication time constraints, the experiments focus mainly on the effect of the computing speed, memory and reliability constraints of CRs, as Eq.10.17. Other information of CRs (e.g., the bandwidths of communication routes in group among CRs and the bandwidths among groups) are generated before allocation and they are all constant value during the solution process. The extraction of the priori knowledge and the effect of constraints would be tested in three cases: $CCR = 1/10$, $CCR = 1$ and $CCR = 10$ in different MTs' DAGs.

More specifically, it assumed that there are 20 available CRs in 4 groups separately, and group. 1 represents the stand-alone CRs group. The quantities of CRs per group are {7, 6, 4, 3}. According to Eq.10.15, the best fitness values in tests are the inverse of the minimum make spans of MTs. So the optimal objection is finding the maximum fitness value. Because the make spans of MTs (in seconds) are usually big, the parameter *Const* in Eq.10.15 is set as 1000 to make the object function results not too small. Then the units of best fitness value in the experiments arems^{-1}. In the algorithm, the maximum time of iteration is set to be 1000, and the population size is set to be 50. A total of 100 runs of each experimental setting are conducted and the average fitness of the best solutions throughout the run is recorded.

Table 10.3 Experiment results with different heuristic parameters

(α, β, γ)	Average minimum make span of MTs (when $CCR = 1$) (units: ms^{-1})			
	$(0.5, 1, 1)$	$(1, 0.5, 1)$	$(1, 1, 0.5)$	$(1, 1, 1)$
ACO	8.4378	8.6126	8.7284	8.5469
IA	8.7347	8.7962	8.8082	8.7593
NDIA	8.8455	8.8863	9.0034	8.8773
(α, β, γ)	Average minimum make span of MTs (when $CCR = 1/10$) (units: ms^{-1})			
	$(0.5, 1, 1)$	$(1, 0.5, 1)$	$(1, 1, 0.5)$	$(1, 1, 1)$
ACO	1.8891	2.0065	1.9658	1.9422
IA	2.0678	2.1012	2.0913	2.0787
NDIA	2.1006	2.1277	2.1201	2.1139
(α, β, γ)	Average minimum make span of MTs (when $CCR = 10$) (units: ms^{-1})			
	$(0.5, 1, 1)$	$(1, 0.5, 1)$	$(1, 1, 0.5)$	$(1, 1, 1)$
ACO	1.456	1.4163	1.4371	1.4299
IA	1.5961	1.5294	1.5481	1.572
NDIA	1.6914	1.6286	1.6392	1.6469

10.7.1 The Design of the Heuristic Information in the Intelligent Algorithms

Owing to GA does not need the heuristic information, in this section, experiments just be carried out on ACO, IA and NDIA. According to Eq. 10.17, choosing a set of suitable parameters α, β and γ is critical in these three algorithms which need the direction of the heuristic information, and in the parameter sets (α, β, γ), low value indicated the low effect in heuristic. Different heuristics may lead different results. With multiple constraints, how to extract suitable heuristic information for better solutions in different situation is very important.

In this experiment, the initial pheromone value of ACO is 1 and its evaporation factor is 0.5. The rates of pheromone and heuristic information in ACO are 1 and 5 separately, and in IA, the crossover and mutation rates are 0.8 and 0.15 separately. Then the initial annealing temperature and its decay factor are 100 and 0.95 separately. Based on the preferences in IA, the rates of pheromone and heuristic information in Eq.10.18 are the same as ACO (i.e. $\varphi = 1$, $\phi = 5$).

Table 10.3 and Fig. 10.12 visualize the effect of different heuristic information on the average minimum make span of MTs figured by the three algorithms. In these experiments, DAG1 is adopted and tests are carried in three situations of CCR.

First, in low communication situation (CCR = 1/10), the set (1, 0.5, 1) can get the best results while the set (0.5, 1, 1) can get the worst, that means, low bandwidth information with high speed and reliability information can guide the algorithms to a better solution, and with low computing speed information, the algorithms are leaded to worse solutions. This is quite reasonable that computing speed is the most important information and bandwidth is the least important one. Because in this situation, computation accounted for larger proportion and then the effect of bandwidth is minor.

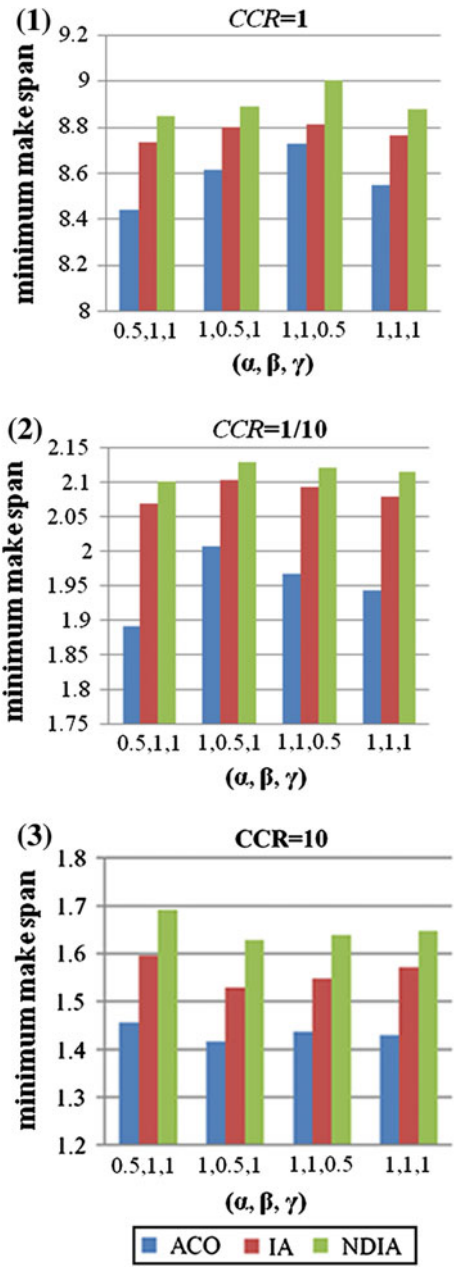


Fig. 10.12 The effect of different heuristic information in OACR

Second, in medium communication situation ($CCR = 1$), it is can be seen that the heuristic with low reliability can get the best results. By now, both computation and communication in MTs are important. Bandwidth and computing speed as their direct influencing factors separately are equally important. So reducing the

rate of the indirect acting factor (the reliability information) and emphasize the other two can promote searching efficiency in algorithms. However, with the same proportion of these three kind of information, worse results would be gotten as the result of the interference of unimportant factor.

Third, in high communication situation ($CCR = 10$), bandwidth information seems to be the most important information with the high proportion of communication in MTs. At this time, low computing speed information can lead a better evolution, and when bandwidth heuristic is lower, the solution is lower, too. Hence the reliability heuristic is also an important factor in this situation. According to the constraints description in Sect. 10.5.3, the reliability factor only effects the communication time when allocating CRs for MTs. So it has large influence in high communication situation and has small influence in low communication situation, just as shown in Fig. 10.12.

Therefore, we can draw a conclusion that the computation speed influences much in low communication situation while the bandwidth and reliability have large effect in high communication situation, but in all of the three situations, equal proportions of three factors could not obtain good solutions.

With CCR $= 1$ and its best parameter set $(\alpha, \beta, \gamma) = (1, 1, 0.5)$, the comparison of the four algorithms (i.e. GA, ACO, IA, NDIA) is carried in the next section.

10.7.2 The Comparison of GA, ACO, IA and NDIA for Addressing OACR

In this experiment, algorithms are tested in three DAGs, and the preferences of GA are the same as IA and NIA, that is $p_c = 0.8$, $p_m = 0.15$.

Figure 10.13 and Table 10.4 show the performance results of the four intelligent algorithms for addressing the OACR problems. The run time, standard deviation, average best fitness, the best solution results and the worst solution results in 100 runs are listed.

(1) Search capability

As shown by results, in precision, NDIA get the best solutions compared with the other three algorithms while IA takes the second place, and the standard GA is the worst. In the aspect of the worst fitness, ACO is the best, and NDIA is the next. In ACO, ants find route from the initiation so their initiate population wouldn't be so bad. The pheromone provides the posterior information to ants to achieve cooperation searching, but it is also easy to make the algorithms trapped into local optimum. So the best solution of ACO is not really good. In NDIA, the niche strategy after the initiation and before the selection increases the diversity of the population. Its good climbing ability makes the algorithm's worst solution in 100 runs better than others. With the incorporation of genetic evolution and niche strategy, the pheromone and dynamic vaccination in NDIA can not only increase the robustness of the heuristics searching, but also avoid the local optimum. So it can always find the best solution compared with other three algorithms.

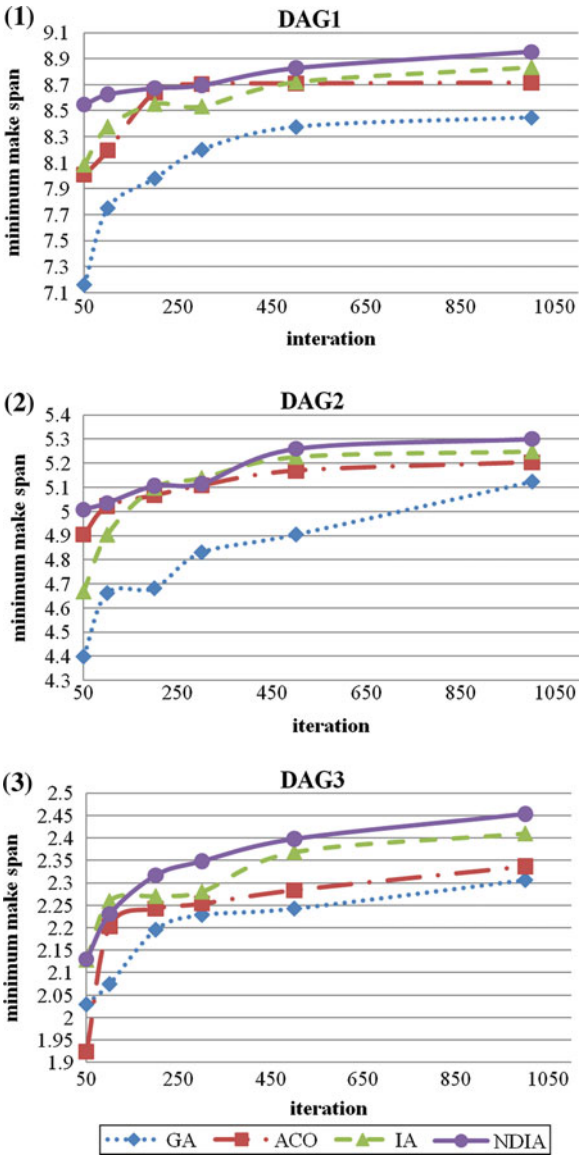


Fig. 10.13 Evolutionary trend of the four intelligent algorithms for addressing OACR

From the climbing ability point of view, GA is the best, NDIA is the next. However, due to the basic stochastic crossover and mutation, GAis easy to trap into local optimum and finally couldn't find the best solutions. On the contrary, NDIA can keep a better evolutionary trend because of its dynamic vaccination strategy. ACO is the worst just because the simple pheromone and heuristic direction cause

Table 10.4 Performance of the four intelligent algorithms for addressing OACR

Graph Number	Algorithms	The worst fitness	The best fitness	Average fitness	Standard deviation	Time
DAG1 (100 runs)	GA	7.5653	9.3467	8.4485	0.4262	163.13
	ACO	8.1071	9.1214	8.716	0.218	3250.44
	IA	7.987	9.5617	8.8352	0.403	226.86
	NDIA	8.0657	9.5617	8.9529	0.3074	811.55
DAG2 (100 runs)	GA	4.6547	5.7213	5.1237	0.2958	255.3
	ACO	5.0828	5.5556	5.2184	0.1313	6731.88
	IA	4.7512	5.8265	5.2483	0.2574	326.61
	NDIA	4.7775	5.9687	5.3008	0.2086	1324.07
DAG3 (100 runs)	GA	1.7043	2.6405	2.3064	0.1688	419.71
	ACO	2.2205	2.6516	2.3367	0.1407	9512.52
	IA	2.046	2.7276	2.41	0.1538	538.28
	NDIA	2.1535	2.7791	2.4533	0.1371	2019.99

the ants gathered quickly into a local optimal solution.And based on dynamic IA's evolution, the niche strategy eliminates the similar individuals and keeps the population searching new area. Thus the climbing ability of NDIA is quite good.

(2) Stability

From Table 10.4 it can be seen that, GA's convergence speed is slow and its stability is the worst of all. Based on the genetic strategy, IA is the next. The initiation in genetic is totally stochastic without heuristic. The evolutions in certain generations are not very stable. ACO's convergence rate is quite good. Because of the pheromone and heuristic, ants can always gather quickly to some extent. With the constant initial pheromone and heuristic, the initial paths founded by ants are fairly stable. Thus the fast convergence and stable initiation makes ACO the most stable algorithm in solving OACR. In NDIA, with the injection of pheromone, dynamic vaccination in IA could be more stable like ACO, and the ability of skipping the local optimum from the niche strategy makes it less stable than ACO Table 10.5.

(3) Time consuming

From the testing results it is clear that ACO is the most time consuming algorithm. According to the analysis in Sect. 10.5.3, ACO needs to compute the heuristic values for all of CRs in every iteration, the complexity of ants' routing is O(*mnv*). With limited MTs, CRs and populations, ACO is the most complex one compared with the other three, and with the addition of pheromone updating and niche strategy, NDIA is more time consuming than IA and GA. However, the complexity of NDIA does not increase significantly in theory, just as shown in Table 10.2.

From the global perspective, NDIA showed high performance in all scales of MTs in OACR. Niched strategy improved the algorithm's exploration and the introduction of experiential pheromone and dynamic heuristics improved the

Table 10.5 Testing results of different parallel methods in solving OACR

Mode\Proc_num	Average_fitness(1000 s^{-1})								
	1	2	3	6	9	12	15	18	21
Independent	7.6968	7.4651	7.2909	6.5734	6.9808	6.3716	6.3622	5.8155	5.6282
Ring	7.6968	7.3982	7.3849	7.4501	7.3661	7.3693	7.4034	7.3637	7.3031
Neighbor	7.6968	7.3724	7.405	7.4985	7.3782	7.4081	7.3625	7.4506	7.3154
Gridsingle	7.6968	7.3951	7.4704	7.5114	7.4066	7.4228	7.3705	7.3958	7.3265
Griddouble	7.6968	7.5106	7.4436	7.5239	7.4735	7.4859	7.3992	7.4293	7.3917
Total	7.6968	7.4814	7.5039	7.4241	7.5468	7.4285	7.4339	7.4626	7.3521
Random	7.6968	7.4287	7.5428	7.4639	7.5213	7.5036	7.4761	7.378	7.4267
Master	7.6968	7.2723	7.3367	7.3523	7.4225	7.2896	7.319	7.2639	7.2578
New	7.6968	7.4384	7.4761	7.5032	7.4862	7.5302	7.4497	7.5216	7.4539
Mode\Proc_num	Average_time(s)								
	1	2	3	6	9	12	15	18	21
Independent	23.594	6.4976	3.7991	2.0284	1.6629	1.4305	1.0965	1.0004	0.9519
Ring	23.594	10.547	7.6201	5.3207	4.4047	3.8511	3.6547	3.6573	3.3041
Neighbor	23.594	10.794	7.8972	5.5503	4.7451	4.1131	3.7188	3.6104	3.3748
Gridsingle	23.594	10.7703	7.4568	5.0895	4.2516	3.6961	3.4102	3.5991	3.2851
Griddouble	23.594	10.1286	7.684	5.1871	4.3461	3.9787	3.3344	3.6379	3.2632
Total	23.594	10.3114	7.0071	4.4832	4.2734	4.1316	3.3544	3.2454	3.0284
Random	23.594	11.0542	9.3598	5.6021	4.6448	4.1999	4.7577	5.1886	5.7958
Master	23.594	12.1286	7.932	8.8628	8.7232	8.7056	8.8172	9.5644	9.5623
New	23.594	10.7358	6.9579	5.1083	4.1406	3.6668	3.2832	3.2339	2.8566

algorithm's exploitation. NDIA got a better balance between exploration and exploitation by these two strategies for addressing OACR in CMfg. From the perspective of solution quality and stability, NDIA has big potential in solving this kind of allocation problem without the increase of time complexity.

10.7.3 The Performance of PNIA

In this section, we mainly apply master-slave, independent, single-ring, double-ring, single-mesh, double-mesh, full-mesh and random topologies mentioned in Chap. 5 based on the configured NIA for addressing OACR problem in CMfg. In the experiments, only DAG3 in Fig. 10.11 is used. For uniformity, all of the above topologies are implemented with 'The best-replace-the worst' migration mechanism and in each period only one individual is migrated. Also, in this test, the total generation number is set to be 2000, *MT* is set to be 20. Other parameters still follow the settings in above section. The parallel environments are one computer with 4-cores and three computing resources with 16-cores in each.

(1) **Time consumption**

According to the tests, the time consumption of independent parallel algorithm (i.e., there is no individual exchange between any computing nodes, the results are received after iteration) is exponentially reduced along with the increase number of sub-processors (i.e. the number of sub-populations). When the number of sub-populations is below to 6, we can get linear speedup directly. When the number is increased further, the time reduction becomes less. The most consumption scheme is master-slave mode. Compared with independent scheme, the time consumption from low to high is: single-ring $<$ double-ring $<$ single-mesh $<$ double-mesh, correspondingly. The difference between them is still small. With MPI_Allgather, full-mesh topology in the experiments performs a little better than the above four topologies.

As the increase of sub-processors, random topology performs well when the processors below 6 and bounce back again when the processors continue to increase. The large time consumption of random topology in the case of large processors mainly ascribes to the production and broadcasting of random control matrix in each period. The point-to-point communication mechanism is also partly responsible for the large time consumption when the processors are continue to increase.

To see the performance of the new configured adaptive full-mesh mechanism, we could find that before the processors come to 9, the time consumption is near to the general full-mesh topology. When the number of processors is larger than 9, the time consumption of it is significantly reduced. In the case with 21 processors, its time consumption becomes the minimum. Therefore, compared with the general full-mesh topology, the adaptive mechanism, just as analyzed above, can effectively reduce the communication load.

(2) **Solution quality**

From the perspective of solution quality, the independent one without any collaboration gets the worst solution result. For the reason that in each sub-population, less individuals is much powerless without the communication with others. Among ring topology and mesh topology, their solution quality from good to bad can be listed as: single-ring < double-ring < single-mesh < double-mesh. During the whole test, single-ring topology shows obviously low searching capability in the specific problem. With the increase of individual exchange in the whole population, we can see that the searching capability of mesh-topology mechanisms is better than that of ring-topology mechanisms. Likely, double-side exchange scheme performs always better than single-side exchange for solving OACR problem. Moreover, from the solution stability point of view, single-ring topology with lower communication and high diversity is the most unstable one of the four schemes. In contrast, double-mesh with the most collaboration is quite stable than the others.

Compared with the four schemes, full-mesh topology performs slightly better. But its stability is worse than double-grid topology. High communication especially when more processors are adopted makes the whole population have low diversity and is partly responsible for its low stability. As we analyzed before, too frequent communication is apt to disrupt the searching direction of each sub-population.

Different with the full-mesh one, random topology performs quite well in solving OACR. The overall solution quality keeps a high level near to the full-mesh one. It has high stability along with the increase of processor number. The random collaboration in each period not only makes high diversity in each sub-population, but also avoids disrupting the whole searching pace and preserves good solution quality. Combined with its time performance, it can be seen that random-topology tradesits searching time for solution quality and stability to some extent.

Further, inspired by the general adaptive mechanisms in improved intelligent optimization algorithm, the new adaptive full-mesh topology has better solution capability than the random one. With population supervision, the amount of data connected and broadcasted in each period is much less than which in random-topology. And the population state can better reflect the evolutionary process and guide the individual exchange. Moreover, with MPI_Allgather, which performs more efficient than the other MPI point-to-point schemes, the searching capability of the whole algorithm is improved without the increase of time consumption.

On the whole, the newly presented PNIA with adaptive full-mesh topology shows high performance in solving OACR problem. Its time consumption with 21 processors is 2.1923 s, which is 10 times lower than the searching with single machine. The speed up ratio of it keeps linear when the processor number is below to 4. The solution accuracy is also largely improved to a high level compared with the other traditional topology. And with the full use of the collective communication of MPI, the new configured topology is also easy to implement.

However, we should notice that in different problems and different computing environments, the solution capability of each topology is changing and quite

unstable. The tests above are only focus on solving the OACR problem in CMfg. With changing environment, it does not always work well to other problems.In this case, we can configure other topology module and adopt multiple algorithms if required. That is the advantage of configuration ways. It is also suggested that in such small cluster environment, the improved configuration based on a single scheme performs better than the topology configuration mentioned in Chap. 5. For adjusting changing environments, multiple configured serial intelligent optimization algorithm can also adopted to improve the whole searching efficiency.

10.8 Summary

Optimal allocation of computing resources is one of the most important and basic problem in CMfg. Current works related to the allocation (scheduling) model and algorithms are either unsuitable or inefficient. This chapter presented a new model with considering the characteristics of CMfg thoroughly and then designed a high efficient intelligent algorithm for OACR in CMfg. In detail, the primary works and contribution of this chapter can be concluded as follows.

(1) In the new OACR model, user's interaction (control and supervision) in manufacturing tasks were fully considered. From the computing aspect, dynamic computing speed was presented associated with processor memory. This technique can clearly reflect the characteristics of resource partitioning in virtualization. Then, from the communication aspect, new cache technique for avoiding data surge, local and remote communication and the communication reliability (rate and recovery time) are introducedin the OACR model. With the full consideration of dynamic computation and communication, the process of OACR in CMfg can be more flexible and practical.
(2) For solving the new complex model, an intelligent optimization algorithm (NIA) is configured with the introduction of niche strategy, immune heuristics, genetic operators and pheromone strategy. Experiments demonstrated the design of heuristic information in NIA for OACR and the suitable heuristics in different situations.
(3) For improve the searching efficiency further, a new adaptive population supervision mechanism is designed and configured with full-mesh topology based on coarse-grained parallelization and MPI collective communication. By using the population degradation state, the individual exchange in each period is controlled to preserve the solution quality with less time consumption. Compared with traditional serial intelligent algorithms and classical parallel intelligent algorithms respectively, the searching capability and time efficiency of the new PNIA was fairly remarkable in the experiments for OACR in CMfg.

References

1. Li BH, Zhang L, Wang SL, Tao F, Cao JW, Jiang XD, Song X, Chai XD (2010) Cloud manufacturing: a new service-oriented networked manufacturing model. Comput Integr Manuf Syst 16(1):1–16
2. Laili YJ, Tao F, Zhang L, Sarker BR (2012) A study of optimal allocation of computing resources in cloud manufacturing systems. Int J Adv Manuf Technol 63(5–8):671–690
3. Yusuf YY, Sarhadi M, Gunasekaran A (1999) Agile manufacturing: The drivers, concepts and attributed. Int J Prod Econ 62(1–2):33–43
4. Flammia G (2001) Application service providers: challenges and opportunities. IEEE Intell Syst Appl 16(1):22–23
5. Tao F, Hu YF, Zhou ZD (2008) Study on manufacturing grid & its resource service optimal-selection system. Int J Adv Manuf Technol 37(9–10):1022–1041
6. Tao F, Zhang L, Venkatesh VC, Luo YL, Cheng Y (2011) Cloud manufacturing: a computing and service-oriented manufacturing model. Proc Inst Mech Eng, Part B, J Eng Manuf (2011, March 10, Accepted)
7. Zhang L, Luo LY, Tao F, Ren L, Guo H (2010) Key technologies for the construction of manufacturing cloud. Comput Integr Manuf Syst 16(11):2510–2520
8. He K, Zhao Y (2005) Research of grid resource management and scheduling. J WuHan Univ Technol (Information and Management Engineering) 27(4): 1–5
9. Ullman JD (1975) NP-complete scheduling problems. J Comput Syst Sci 10(3):384–393
10. Zhang L, Luo YL, Fan WH, Tao F, Ren L (2011) Analysis of cloud manufacturing and related advanced manufacturing models. Comput Integr Manuf Syst 17(3):458–468
11. Li BH, Zhang L, Chai XD, Tao F, Luo YL, Wang YZ, Yin C, Huang G, Zhao XP (2011) Further discussion on cloud manufacturing. Comput Integr Manuf Syst 27(3):449–457
12. Tao F, Cheng Y, Zhang L, Luo YL, Ren L (2011) Cloud manufacturing. The 2nd international conference on manufacturing service and engineering (ICMSE)
13. Tao F, Zhang L, Luo YL, Ren L (2011) Typical characteristic of cloud manufacturing and several key issues of cloud service composition. Comput Integr Manuf Syst 17(3):477–486
14. Liang JJ, Pan QK, Chen TJ, Wang L (2011) Solving the blocking flow shop scheduling problem by a dynamic multi-swarm particle swarm optimizer. Int J Adv Manuf Technol 55(5–8):755–762
15. Zou ZM, Li CX (2006) Integrated and events-oriented job shop scheduling. Int J Adv Manuf Technol 29(5–6):551–556
16. Hu PC (2005) Minimizing total flow time for the worker assignment scheduling problem in the identical parallel-machine models. Int J Adv Manuf Technol 25(9–10):1046–1052
17. Kwok YK (1999) Benchmarking and comparison of the task graph scheduling algorithms. J Parallel Distrib Comput 59(3):381–422
18. Polychronopoulos CD (1991) The hierarchical task graph and its use in auto-scheduling. In: Proceedings of the 5th international conference on supercomputing (ICS' 91)
19. Bokhari SH (1979) Dual processor scheduling with dynamic reassignment. IEEE Trans Software Eng 5(4):341–349
20. Stone HS (1977) Multiprocessor scheduling with the aid of network flow algorithms. IEEE Trans Software Eng 3(1):85–93
21. Madhukar M, Leuze V, Dowdy V (1995) Petri net model of a dynamically partitioned multiprocessors system. In: Proceedings of the 6th international workshop on petri nets and performance models (PNPM' 95)
22. Buyya R, Abramson D, Venugopal S (2005) The grid economy. Proc IEEE 93(3):698–714
23. Cardoso J, Sheth A, Miller J, Arnold J, Kochut K (2004) Quality of service for workflows and web service processes. Web Semant: Sci Serv Agents WWW 1(3):281–308
24. Yang T, Gerasoulis A (1993) DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Trans Parallel Distrib Syst 5(9):951–967

25. Gerasoulis A, Yang T (1993) On the granularity and clustering of directed acyclic task graphs. IEEE Trans Parallel Distrib Syst 4(6):686–701
26. Gerasoulis A, Yang T (1994) Performance bounds for parallelizing Gaussian-Elimination and Gauss-Jordan on message-passing machines. Applied Numerical Mathematics Journal 16:283–297
27. Jones WM, Pang LW, Ligon WB, Stanzione D (2005) Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. J Supercomput 34(2):135–163
28. Hamscher V, Schwiegelshohn U, Streit A, Yahyapour V (2004) Evaluation of job-scheduling strategies for grid computing. Grid Computing at the 7th International Conference on High Performance Computing 191–202
29. Ememann C, Hamscher V, Yahyapou V (2002) On effects of machine configurations on parallel job scheduling in computational grids. In: Proceedings of the international conference on architecture of computing systems (ARCS 2002), 169–179
30. Davidovi T, Hansen P, Mladenovi N (2005) Permutation based genetic, tabu and variable neighborhood search heuristics for multiprocessor scheduling with communication delays. Asia-Pac J Oper Res 22(3):297–326
31. Sinnen O, Sousa LA (2005) Communication contention in task scheduling. IEEE Trans Parallel Distrib Syst 16(6):503–515
32. Sinnen O, Sousa LA, Sandnes FE (2006) Toward a realistic task scheduling model. IEEE Trans Parallel Distrib Syst 17(3):263–275
33. Benoit A, Marchal L, Pineau JF (2010) Scheduling concurrent bag-of-tasks applications on heterogeneous platforms. IEEE Trans Comput 59(2):202–217
34. Adam TL, Chandy KM, Dickson JR (1974) A comparison of list schedules for parallel processing systems. Commun ACM 17(12):685–690
35. Sinnen O, Sousa LA (2004) List scheduling: Extension for contention awareness and evaluation of node priorites for heterogeneous cluster architectures. Parallel Comput 30(1):81–101
36. Wu MY, Gajski DD (1990) Hypertool: a programming aid for message-passing systems. IEEE Trans Parallel Distrib Syst 1(3):330–343
37. Sarkar V (1989) Partitioning and scheduling of parallel programs for multiprocessors. Research Monographs in Parallel Computing, MIT Press
38. Chen S, Eshaghia MM, Wu Y (1995) Mapping arbitrary non-uniform task graphs onto arbitrary non-uniform system graphs. In: Proceedings of the international conference on parallel processing
39. Yang L, Gohad T, Ghosh P, Sinha D, Sen D, Richa A (2005) Resource mapping and scheduling for heterogeneous network processor systems. In: Proceedings of the 2005 ACM Symposium on Architecture for Networking and Communications Systems *(ANCS' 05)*, 19–28
40. Weng N, Wolf T (2005) Profiling and mapping of parallel workloads on network processors. In: proceedings of the 20th annual ACM symposium on applied computing (sac) 890–896
41. Huang JG, Chen JE, Chen SQ (2004) Parallel-job scheduling on cluster computing system. Chin J Comput 27(6):765–771
42. Huang JG (2008) Approximation algorithm on multi-processor job scheduling. Comput Eng Appl 44(32):26–28
43. Yin GF, Luo Y, Long HN, Cheng EJ (2004) Genetic algorithms for subtask scheduling in concurrent design. J Comput aided Des Comput Graph 16(8): 1122–1126
44. Correa RC, Ferreira A, Rebreyend P (1999) Scheduling multiprocessor tasks with genetic algorithms. IEEE Trans Parallel Distrib Syst 10(8):825–837
45. Tsai JT, Liu TK, Ho WH, Chou JH (2008) An improved genetic algorithm for job-shop scheduling problems using taguchi-based crossover. Int J Adv Manuf Technol 38(9–10):987–994

46. Chen YW, Lu YZ, Yang GK (2008) Hybrid evolutionary algorithm with marriage of genetic algorithm and extremal optimization for production scheduling. Int J Adv Manuf Technol 36(9–10):959–968
47. Wang G, Gong WR, DeRenzi B, Kastner R (2007) Ant colony optimizations for resource and timing constrained operation scheduling. IEEE Trans Comput Aided Des Integr Circuits Syst 26(6):1010–1029
48. Chen WN, Zhang J (2009) An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. IEEE Trans Syst Man Cyber 39(1):29–43
49. Li JQ, Pan QK, Gao KZ (2011) Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. Int J Adv Manuf Technol 55(9–12):1159–1169
50. Xu XD, Li CX (2007) Research on immune genetic algorithm for solving the job-shop scheduling problem. Int J Adv Manuf Technol 34(7–8):783–789
51. Agarwal R, Tiwari MK, Mukherjee SK (2007) Artificial immune system based approach for solving resource constraint project scheduling problem. Int J Adv Manuf Technol 34(5–6):584–593
52. Saravanan M, Haq AN (2008) Evaluation of scatter-search approach for scheduling optimization of flexible manufacturing systems. Int J Adv Manuf Technol 38(9–10):978–986
53. Laha D, Chakraborty UK (2008) An efficient heuristic approach to total flowtime minimization in permutation flow shop scheduling. Int J Adv Manuf Technol 38(9–10):1018–1025
54. Maheswaran R, Ponnambalam SG, Aravindan C (2005) A meta-heuristic approach to single machine scheduling problems. Int J Adv Manuf Technol 25(7–8):772–776
55. Zhang JX, Gu ZM, Zheng C (2010) Survey of research progress on cloud computing. Appl Research Comput 27(2): 429–433
56. Hong B, Prasanna VK (2004) Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. In: Proceedings of the 18th international parallel and distributed processing symposium (IPDPS' 04)
57. Bhat PB, Raghavendra CS, Prasanna VK (2003) Efficient collective communication in distributed heterogeneous systems. J Parallel Distrib Comput 63(3):251–263
58. Gawiejnowics S (2008) Time-dependent scheduling. Springer, Berlin
59. Wang L, Pan J, Jiao LC (2000) The immune programming. Chin J Comput 23(8): 806–812
60. Wang L, Pan J, Jiao LC (2000). The immune algorithm. Acta Electronica Sinica, 28(7): 74–77
61. Park J, Kang M, Lee K (1996) An intelligent operations scheduling system in a job shop. Int J Adv Manuf Technol 11(2):111–119
62. Jiao LM, Khoo LP, Chen CH (2004) An intelligent concurrent design task planner for manufacturing system. Int J Adv Manuf Technol 23(9–10):672–681
63. Chaudhry IA, Drake PR (2009) Minimizing total tardiness for the machine scheduling and worker assignment problems in identical parallel machines using genetic algorithms. Int J Adv Manuf Technol 42(5–6):581–594
64. Saravanan M, Haq AN (2008) Evaluation of scatter-search approach for scheduling optimization of flexible manufacturing systems. Int J Adv Manuf Technol 38(9–10):978–986
65. Wang LY, Wang JB, Gao WJ, Huang X, Feng EM (2010) Two single-machine scheduling problems with the effects of deterioration and learning. Int J Adv Manuf Technol 46(5–8):715–720
66. Jerald J, Asokan P, Saravanan R, Delphin R, Rani C (2006) Simultaneous scheduling of parts and automated guided vehicles in an FMS environment using adaptive genetic algorithm. Int J Adv Manuf Technol 29(5–6):584–589
67. Shukla SK, Son YJ, Tiwari MK (2008) Fuzzy-based adaptive sample-sort simulated annealing for resource-constrained project scheduling. Int J Adv Manuf Technol 36(9–10):982–995

Chapter 11
Job Shop Scheduling with FPGA-Based F4SA

In this chapter, a new configured permutation-based feasible solution space searching simulated annealing algorithm (F4SA) is designed for solving job shop scheduling problem (JSSP). Firstly, a permutation-based encoding scheme is presented, which can make the solution always feasible in iteration. After that, simulated annealing operator, mutation operator and a new reverse order operator are implemented on FPGA and configured for updating solutions in parallel way. Each operator is encapsulated in a module and can be connected with fixed input, output and parameters. The searching time of intelligent optimization algorithm in FPGA is far shorter than which in general computer. The design and implementation of F4SA for JSSP presented in this chapter is just an example to demonstrate how to implement an intelligent optimization algorithm and dynamically configure multiple operators in FPGA. The searching accuracy of this algorithm is to be improved further.

11.1 Introduction

Job shop scheduling problem (JSSP) is an important practical problem in manufacturing system which directly decides the efficiency of manufacturing process. It is also a very important theoretical problem due to its complexity. Thus it has been widely studied for more than half a century.

The literatures on JSSP are quite a lot. Many exact methods have been proposed to find the optimal solutions for JSSP, such as typical branch and bound algorithm presented by Carlier and Pinson [1] which solved FT10 benchmark problem and dynamic algorithm brought up by Lawler [2]. Because JSSP is an NP-hard problem [3], the problem solution space will expand explosively with the growth of the problem scale. Therefore, the exact methods are not guaranteed to solve large size JSSPs in a certain limited time. In this situation, some approximate

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_11

methods are proposed to find suboptimal solutions instead of optimal ones. These methods include dispatching priority rules [4], shifting bottleneck approach proposed [5], Lagrangian relaxation [6], and tree searching algorithm in [7]. These methods have made remarkable achievements. In recent years, intelligent optimization algorithm have been widely studied and applied to solve the problem and genetic algorithm (GA) is the most popular algorithm among them. For instance, Gonçalves et al. [8] brought up a hybrid genetic algorithm to minimize the makespan of JSSP and has achieved good results. Zhou et al. [9] proposed a hybrid genetic algorithm to minimize weighted tardiness of JSSP, and Giovanni and Pezzella [10] improved the GA for the flexible job shop scheduling problem. There are also several other intelligent optimization algorithms applied to solve JSSP except GA, such as particle swarm optimization algorithm (PSO) [11], Kasemset et al. [12], simulated annealing (SA) [13, 14] and differential evolution (DE) [15, 16]. Most of them try to find good solutions among the problem's whole search space, while some of the found solutions are not feasible, which reduces the efficiency of the algorithms. This is due to the JSSP representation model and the encoding scheme.

JSSP can be represented by a disjunctive graph model [17] and permutation model [18]. Each solution, feasible or not, is covered in the search space in the disjunctive graph model. The solution will be rebuilt when it is not feasible, and GT algorithm [19] proposed by Giffler, and Thompson is a common used builder. After rebuilding, the new solution will meet the problem's constraints. It should be pointed that the rebuilding procedure will take a certain amount of time, which slows down the decision process of algorithm.

In the permutation model, the processing sequence of jobs on each machine is encoded as a permutation of numbers. If the solutions are encoded in random permutations, they will also cover the problem's whole search space. Several kinds of intelligent optimization algorithms are applied to the model with different encoding scheme. For example, differential evolution proposed by Ponsich et al. [20] with permutation model in two different encoding schemes is applied to solve the problem. But it gives no better results. Artificial immune algorithm (AIA) brought up by Golmakani and Namazi [21] with a dedicated encoding scheme of permutation which only generates feasible solution has achieved very good results.

Clearly, the handling of solution space and generation of feasible solutions are very critical in solving JSSP. Therefore, this chapter focuses on the generation and handling of feasible solutions in JSSP, presents a new permutation-based feasible solution space searching simulated annealing (F4SA) algorithm for it. With the hybridization of simulation annealing rules and multiple transform operators, a novel permutation-based feasible set searching strategy is designed. Compared to previous work, the new feasible set encoding scheme presented in this chapter ensures feasible solutions and the multiple transform operators accelerates evolutions during searching process.

11.2 Problem Description of Job Shop Scheduling

The job shop scheduling problem can be described as follows: a set of n jobs $\{J_i|1 \le i \le n\}$ which is to be processed on a set of m machines $\{M_j|1 \le j \le m\}$, and each job has a technological sequence of machines to be processed. The processing of job J_i on machine M_j is called *operation* O_{ij}. Operation O_{ij} requires the exclusive use of M_j for an uninterrupted duration p_{ij}, i.e. its processing time. A *schedule* is a complete set of operations $S_{m\times n} = \{O_{ij}\ |1 \le i \le n,\ 1 \le j \le m\}$, to be processed on different machines, in a given order.

With the notations defined in the Table 11.1, The conceptual model can be described as following:

Objective:

$$\text{minimize } C_{\max}(S) \tag{11.1}$$

Subject to:

$$t_{i+1,\,j+1} \ge t_{i,\,j+1} + p_{i,\,j+1}, \quad i = 1, 2, \cdots, n, j = 1, 2, \cdots, m \tag{11.2}$$

$$t_{i+1,\,j+1} \ge t_{i+1,\,j} + p_{i+1,\,j}, \quad i = 1, 2, \cdots, n, j = 1, 2, \cdots, m \tag{11.3}$$

Note: $t_{1,j}$ and $t_{i,1}$ are determined by the schedule S, and $i = 1, 2, \cdots, n$, $j = 1, 2, \cdots, m$. We will take 3×3 JSSP to explain the model further.

Table 11.2 shows the technological sequence and the processing time of each operation of a 3×3 JSSP. The objective is to find a schedule of minimal time to complete all jobs. Table 11.3 shows a schedule corresponding to Table 11.2. And Fig. 11.1 shows the Gantt chart of the schedule in Table 11.3.

11.3 Design and Configuration of SA-Based on FPGA

11.3.1 FPGA-Based F4SA Design for JSSP

(1) Encoding scheme

For generating feasible solutions of JSSP directly in both the initialization and the evolutionary process, a new encoding scheme is presented in this chapter. Specifically, a solution can be represented by a $m \times n$ coding sequence, which contains n subsequences with m elements in each. It is also a permutation of 1 to $m \times n$. Take 3×3 JSSP with 3 machines and 3 jobs as an example, the specific decoding way of the permutation sequence is described as follows.

Let the constraints matrix and processing time matrix be

Table 11.1 Conceptual model of JSSP

n	Number of jobs
m	Number of machines
O_{ij}	Operation of number i job on number j machine
t_{ij}	Start time of operation O_{ij}
p_{ij}	Processing time of operation O_{ij}
S	A schedule of complete set of operations
$C_{\max}$	Makespan of schedule S

Table 11.2 Technological constraints and processing time table

Jobs	Number of machines, processing time		
J1	2, 3	3, 7	1, 5
J2	1, 6	3, 9	2, 8
J3	3, 4	1, 3	2, 9

Table 11.3 Schedule table

Machines	Number of jobs		
M1	2	1	3
M2	1	2	3
M3	2	1	3

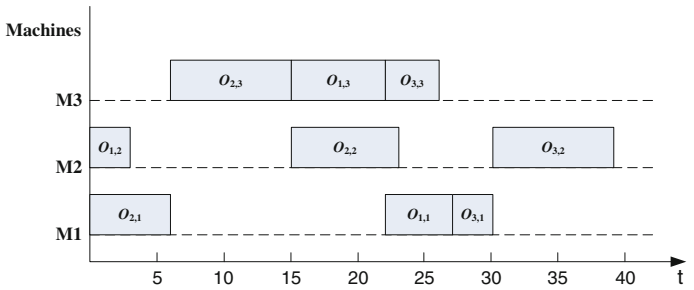


Fig. 11.1 Gantt chart of the schedule shown in Table 11.3

$$Bound = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 3 & 2 \\ 3 & 1 & 2 \end{bmatrix}, \; Time = \begin{bmatrix} 3 & 7 & 5 \\ 6 & 9 & 8 \\ 4 & 3 & 9 \end{bmatrix} \tag{11.4}$$

And let the initial sequence be *Per_ori*, for instance, *Per_ori* = 163528947. Then we need three steps to transform such sequence to a feasible solution of JSSP.

Step 1: Ranking in group

According to the above encoding way, *Per_ori* can be divided into 3 subsequences with 3 elements in each, i.e., *Per_ori* = 163 | 528 | 947. Ranking in group means to sort the elements in each group with ascending order. After the ranking, the sequence can be represented as *Per_sort* = 136 | 258 | 479.

Step 2: Expansion of constraints matrix

In this step, we need a $n \times mn$ middle matrix to expand *Bound* matrix. Let the middle matrix to be *BoundSpread*. Then the value of *BoundSpread* can be calculated according to the following equation.

$$BoundSpan\left(\left[\frac{\mathrm{i}}{n}\right]+1, Per\_sort(\mathrm{i})\right) = Bound\left(\left[\frac{\mathrm{i}}{n}\right]+1, (\mathrm{i}-1)\%\, n+1\right), i = 1,2,\cdots,mn \tag{11.5}$$

Therefore the new generated middle matrix of the above case is:

$$BoundSpread_{n\times mn} = \begin{bmatrix} 2 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 1 & 0 & 2 \end{bmatrix} \tag{11.6}$$

Step 3: Generating schedule

According to the *BoundSpread* matrix, we could see that each machine number from 1 to 3 appears three times. If we check each column and record the row of machine i in the order of appearance, we could get the processing order of each machine. For example, for machine 1, the row number of it in turn are 2, 1 and 3 in column 2, 6 and 7, respectively. Then the schedule order of machine 1 is [2 1 3]. Hence, the corresponding schedule calculated according to the above *BoundSpread* matrix is:

$$Schedule = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 3 \\ 1 & 3 & 2 \end{bmatrix} \tag{11.7}$$

Because the schedule is generated in accordance with the original *Bound* matrix, it must be a feasible solution.

Based on the new encoding scheme, we will design a FPGA-based Decoder for addressing 6×6 JSSP. The external interface of such FPGA module can be shown in Fig. 11.2.

In the decoder module, there are three input port, i.e., clock signal *clk*, enable signal *en* and encoding signal *code*(0:215), and one output port, *schedule*(0:143). The encoding signal is set as 216 bits for solving 6×6 JSSP. Specifically, the coding length of 6×6 JSSP is 36. The maximum value 36 is between 32 and 64. If we adopt unsigned binary number to present the coding, we need at least 6 bits. Therefore, we need 6 bits $\times$ 6 $\times$ 6 = 216 bits. In a similar way, the schedule

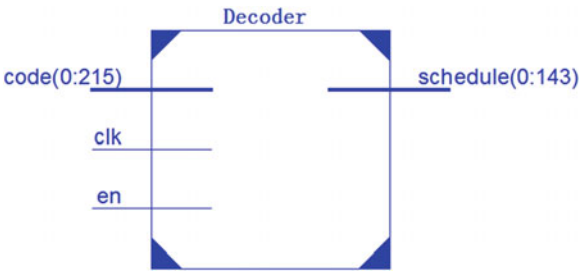


Fig. 11.2 FPGA decoder module for JSSP

signal is set to be 144 bits. That is because each element in schedule is between 1 and 6. We need at least 4 bits to represent it. For the whole schedule, we need 4 bits $\times$ 6 $\times$ 6 = 144 bits. Moreover, en signal is active-low. When the input is high, the code input is sampled.

Considering the decoding process, there are three main steps needed to be implemented in serial. In each step, the operations can be all accelerated by some parallelization way. For instance, ranking of 6 subsequences in step 1 can be directly executed in parallel. In step 2, the calculation of each element in *Boud-Spread* can be parallelized as well. Besides, the column checking in step 3 can also be divided into three independent parallel parts. However, parallelization of all these steps will occupy more logistical resources. Therefore, only step 3 is parallelized here. In this step, we decide the current processing jobs for all 6 machines simultaneously at the rising edge of the clock. The consumption of schedule generation can then be reduced 1/6 of its original time.

In simulation process, the initial value of the encoding signal input *code* is represented by a hexadecimal coding. For instance, if we have the input *code* = "3020C41461C824A2C134E3D04524D45565D865A6DC75E7E08628E4". After decoding, we can get the schedule sequence *schedule* = Ox"123456213456123 456123456213456123456". The loading time of *code* is 300 ns, output time of *schedule* is 9095 ns and the whole processing time of decoding is 9095 − 300 ns = 8795 ns. The simulation clock is 100 MHz, so that it needs 879 clock periods to finish a decoding process.

As we know, the scale of the solution space of JSSP is $(n!)^m$. In such encoding scheme, the solution space will be expanded to $\frac{(mn)!}{(m!)^n} > (n!)^m$. It means that although we can always get a feasible solution from a permutation, this encoding scheme is a many-to-one scheme. $m \times n$ JSSP in such scheme can be transformed as $m \times n$ TSP (Travelling Salesman Problem) with less solution space than that of a real TSP, i.e., $(mn)!$

From the analysis, we can directly apply the permutation operations used in TSP to handle JSSP. The only difference between them are the encoding and decoding process and the evaluation way. For efficiently obtain optimal solution in such many-to-one encoding scheme, a new problem solving scheme is presented based on FPGA implementation framework.

(2) FPGA-based new problem solving scheme

In general, the total processing time of jobs is decided by a feasible schedule *Schedule*, a constraints matrix *Bound* and a processing time matrix *Time*. The calculation process is carried by a FPGA module. In this module, there are three input signals, encoding signal *code*(0:215), clock signal *clk* and enable signal *en*, as well as the decoding module. There is also only one output signal, that is processing time signal *makespan*(7:0). A decoder is imbedded in this module and connect with its three input signals. The output signal *makespan* is a 8 bits unsigned binary *std_logic_vector*, which can represent the integers from 0 to 255. Then the calculation of job processing time in FPGA can be implemented by some state machines as shown in Fig. 11.3.

The above state machines can be directly implemented by VHDL programming. Set code = 0x"3020C41461C824A2C134E3D04524D45565D865A6DC75E7E08628E4". We could get the output result of *CalcMakespan* module $0\times8C = 140$. In simulation, the loading time of *code* signal is 200 ns, the output time of *makespan* is 16505 ns. Therefore the calculation time is $16505 - 200$ ns $= 16305$ ns. The simulation clock is still 100 Mhz, so that it has taken 1630 clock periods to finish the evaluation of a feasible solution. With a decoder module imbedded in *CalcMakespan*, the execution of the whole state machines spent $1630 - 879 = 751$ clock periods, which is slightly shorter than the decoding time.

11.3.2 FPGA-Based Operators of F4SA

The main operator of SA is its acceptance rule. By accepting some bad solutions in probability, the diversity of the population can be improved and wider range of space can be searched as well. With SA acceptance scheme, the searching process of JSSP in FPGA can be demonstrated as Fig. 11.4.

In Fig. 11.4, *tf* represents the terminated temperature, *t*0 refers to the initial temperature, *t* is set as the current temperature, *alpha* stands for the cooling coefficient, *L* is the number of coding update between two cooling steps and *i* is the iterative variable.

After the initialization step, a new solution is generated from the original one. According to the evaluation module described above, we could get its processing time as the new fitness value *makespan_new*. If the value *makespan_new* is smaller than that of the original solution, i.e. *makespan_old*, or $\exp\left(-\frac{makespan\_new - makespan\_old}{t}\right)$ is larger than a temporarily generated uniform random number, then the old solution will be replaced by the new one. If the solution is changed, then $i = i + 1$. When $i \geq L$, set $t = alpha \times t$ and $i = 0$. The iteration then continue until $t < tf$.

For generating new solution, we need to design some operators during the process and implement them as modules on FPGA. In this chapter, the simulated annealing operator, mutation operator and reverse operator are designed and hybrid on FPGA with a new encapsulation way.

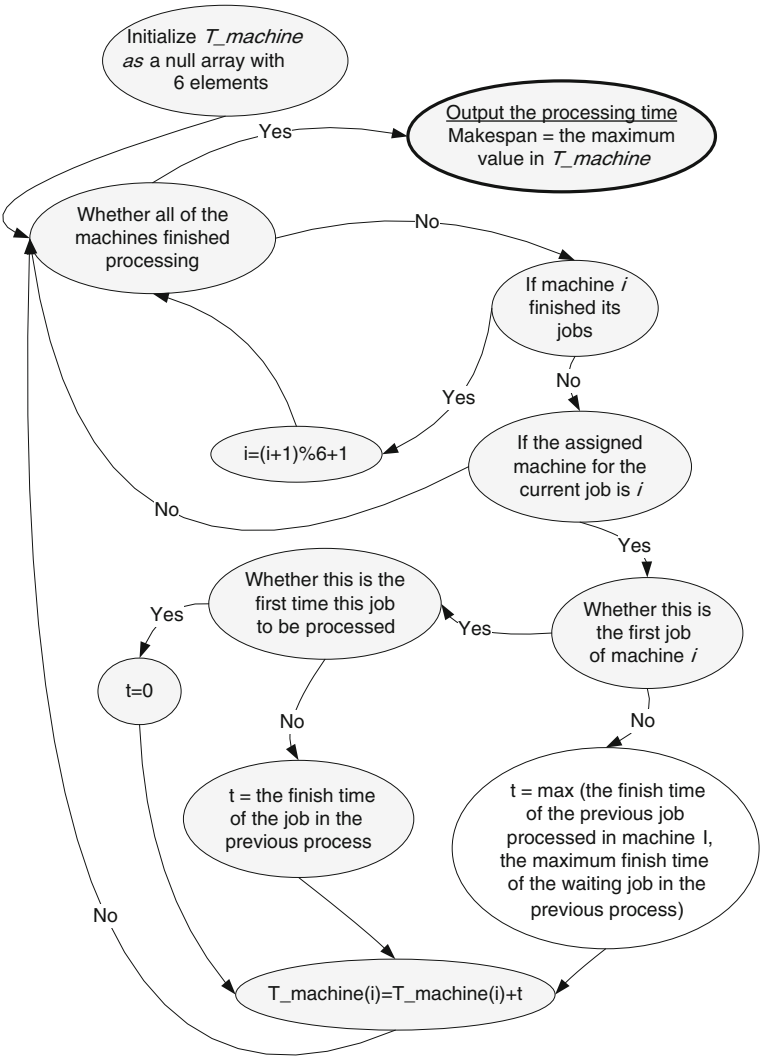


Fig. 11.3 The calculation of job processing time in FPGA

(1) Simulated annealing operator based on FPGA
In solution acceptance rule, we need to calculate the value of exp(x). In FPGA, we adopt the traditional Taylor series expansion for the calculation of acceptance probability.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \cdots \tag{11.8}$$

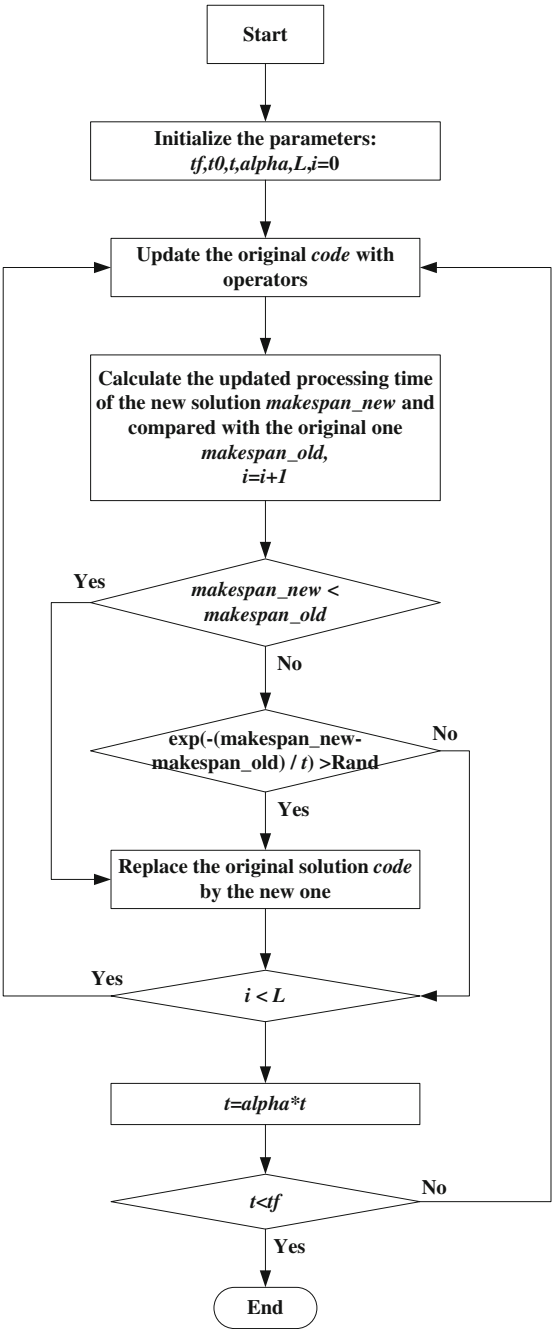


Fig. 11.4 The searching process of FPGA-based SA

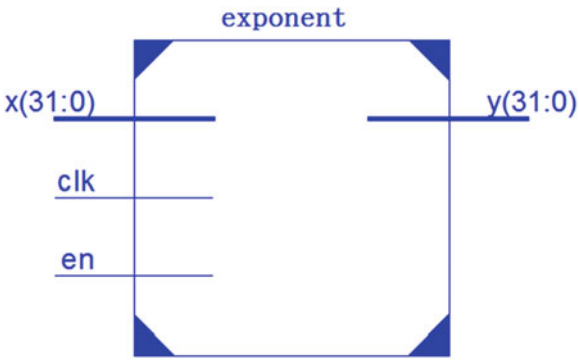


Fig. 11.5 The calculation module of e^x

In the above equation, $x < 0$, so that $e^x < 1$. If we calculate all of the top ten items, the value of e^x can be accurate to 7 decimal places. The calculation module of e^x can then be shown in Fig. 11.5.

The inputs of this module contains the variable $x(31:0)$, clock signal *clk* and enable signal *en*. The output value is represented as y(31:0). x and y are both single precision floating point numbers with IEEE754 standard. The length of both x and y is 32 bits. When the input signal x is 0.9 = 0X3f666666, the simulation result is 0X401D6A22. Compared with the result 0X401D6A23 calculated in Matlab, the accuracy of the calculation module of e^x can be fully assured.

(2) Mutation operator based on FPGA

Mutation operator in the searching workflow is the simplest module in FPGA. It only needs to randomly exchange the value between two bits in the solution *code*. Specifically, we could randomly generate two integer numbers which indicates the switch points, and then exchange them to generate a new solution. In the step of random number generation, we use Matlab to generate a group of uniform random integer number, transfer them into single precision floating point binary number and stored them in the ROM of FPGA. During the iteration, these numbers are then adopted for determining the mutation points.

(3) Reverse order operator based on FPGA

Reverse order operation refers to choose two points in the solution *code*, and reverse the whole subsequence between the two points to generate a new solution. Specifically, let the two randomly chosen points as p_1 and p_2, and set *ps_old* and *ps_new* be the original solution and the new solution respectively, the value of *ps_new* between p_1 and p_2 can be calculated as follow.

$$ps\_new[i] = ps\_old[p_2 + p_1 - i], \quad i = p_1, p_1 + 1, \ldots, p_2 \tag{11.10}$$

For 6 × 6 JSSP, the length of *ps* is 36. For simplicity, assuming the length of *ps* to be 10, as shown in Fig. 11.6. If the input sequence *ps_old* = "0123456789",

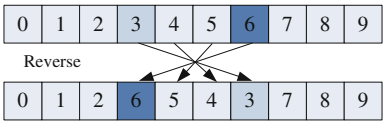


Fig. 11.6 The execution of reverse order operator for a sequence

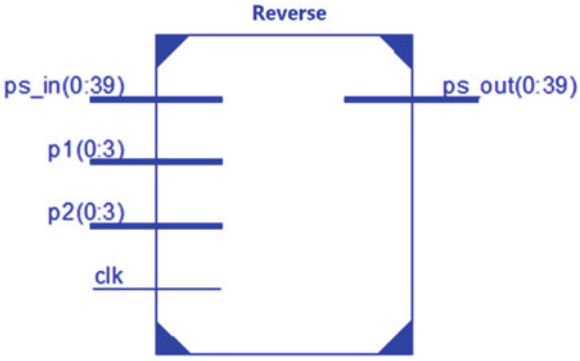


Fig. 11.7 The design module of reverse order operator in FPGA

and $p_1 = 3$, $p_2 = 6$, after the reverse operation, the new sequence ps_new = "0126453789".

The design module of reverse order operator in FPGA can be shown in Fig. 11.7. Except the clock signal, all the inputs are designed corresponding to the variable described above. With binary format, the input values in FPGA can be represented as: ps_in(x "0123456789") = b"00000001 0010 0011 0100 0101 0110 0111 1000 1001", $p1$(x"3") = b"0011", $p2$(x"6") = b"0110", ps_out(x"0126543789") = b"00000001 0010 0110 0101 0100 0011 0111 1000 1001".

Different with mutation operator, the random number p_1 and p_2 are generated by an M sequence generator. For increasing the cycle time of M sequence and enhancing the randomness, in this chapter we set the length of M sequence to be 32. Therefore its cycle time in theory can be $2^{32} = 42.9 \times 10^9$ clock periods. The specific logistic structure of M sequence generator can be shown in Fig. 11.8.

In the logistical structure, the XOR of m_1 and m_0 is fed back as the input of m_{31}. The value b"$m_3m_2m_1m_0$" is then bounded as the output. Before the output, if b"$m_3m_2m_1m_0$" $\geq$ b"1010", then bounded the value by adding b"0110". In this way, the output value can also be bounded in the interval [b"0000", b"1001"] . Hence, the external interface of M sequence generator can be shown in Fig. 11.9.

In this sub-module, init(31:0) get the input b"$m_{31}m_{30}m_{29}\cdot\cdot m_1m_0$" as the initial value. If $reset = 0$, the initial value is loaded into the module asynchronously. After module execution, Mdata(3:0) will output the bounded value of b"$m_3m_2m_1m_0$" for reverse order operator module.

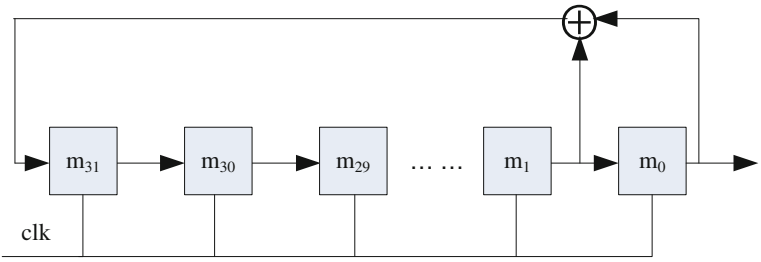


Fig. 11.8 The logistic structure of 32 bits M sequence generator for reverse order operator

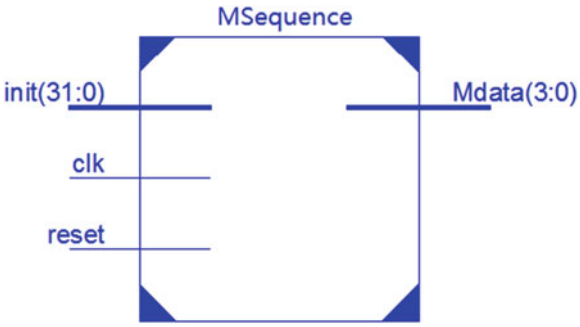


Fig. 11.9 The external interface of M sequence generator

11.3.3 Operator Configuration Based on FPGA

The general configuration process of the above algorithm in FPGA can be shown in Fig. 11.10. In the process, we could not only use the module composition with both reverse operator and simulated annealing operator connected by red full line, but also use the module of only reverse operator with blue dash line, or the module composition with mutation and reverse operators further. Different scheme can be generated for different cases. In each scheme, the candidate solution is input to *CalcMakespan* for evaluation and updating. With experiments, we found that the scheme with both reverse operator and simulated annealing operator performed better than the one with only reverse operator in solving general JSSP. The testing results will be discussed in the following section.

11.4 Experiments and Discussions

In the experiments, the terminal temperature *tf* is set as 0.01, the initial temperature *t* is 500 and the cooling coefficient *alpha* and *L* are set as 0.8 and 360 respectively. Because the uniform random number for simulated annealing operator is pre-generated in FPGA, it does not change in each run. For solving this problem, we

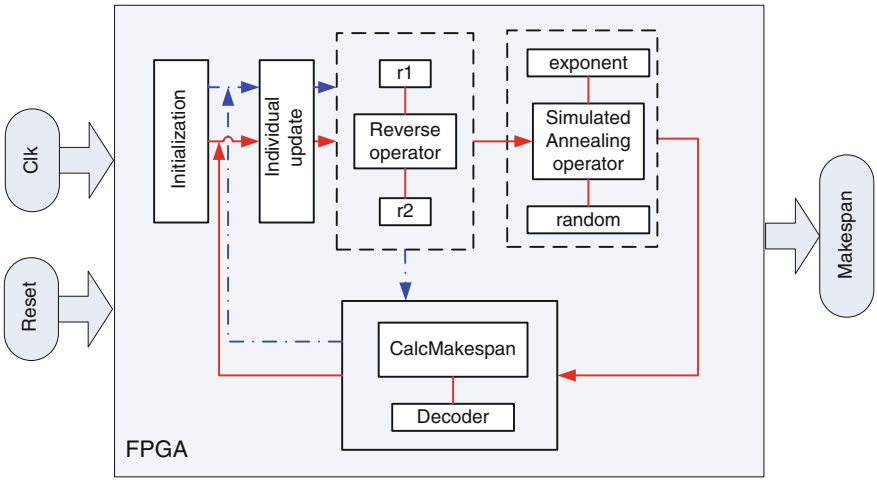


Fig. 11.10 The general configuration process of the improved SA in FPGA

could load different initial numbers for M sequence generator to control the random number generation and make the searching process more dynamic in different runs. In reverse order operator, two random points are needed to be generated in *code*, so that two M sequence generators are required as well. In the experiments, two of them are both 32 bits. For example, if we load the initial numbers 0×12345678 and 0×87654321 for two of the M sequence generators respectively, we can then increase them by 1 after each cooling process is finished and start a new process again. The simulation results can be shown in Fig. 11.11.

It can be seen that with 6 times cooling process, the processing times obtained are 0×41, $0 \times 3C$, $0 \times 3D$, $0 \times 3F$, $0 \times 3D$ and $0 \times 3C$, i.e. 65, 60, 61, 63, 61 and 60, respectively. The processing time of the initial solution in the test is $0 \times 8C$, i.e. 140. Clearly, the processing time is largely reduced. But considering the minimal processing time of this problem, i.e. 55, has not been founded, more slower cooling process may be required.

From the perspective of the searching time in FPGA, we could find that a single cooling process spends only 0.45 s. With the same parameter setting, the process executed by Matlab requires at least 2.07 s. Note that the computer for running Matlab is Lenovo G460 with 32 bits Windows 7 operation system, Intel® Core™ i3, 2 GB memory and 2.53 GHz dominant frequency. In FPGA, the dominant frequency is only 100 MHz. With much lower dominant frequency, FPGA-based intelligent optimization algorithm is much more efficient than the general CPU-based ones. This is in part because the internal operations are parallelized in FPGA-modules. Moreover, the execution process in FPGA contains only the related operators with clock period, while the execution process in CPU consists of not only the operators but also the handling of processors and threads in system. On the

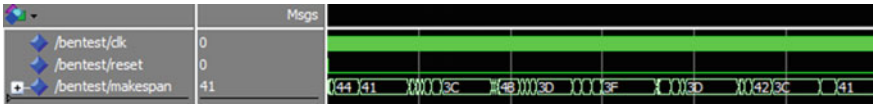


Fig. 11.11 Simulation results of the new improved SA in FPGA for solving JSSP

whole, the design of intelligent optimization algorithm based on FPGA is light-weight with high efficiency, which can fully satisfy the demand of many dynamic light systems.

11.5 Summary

In this chapter, we elaborated the design of F4SA in FPGA. For solving JSSP, simulated annealing operator, mutation operator and reverse order operator are combined and implemented in FPGA. Through these independent modules, an hybrid intelligent optimization algorithm can be dynamically configured on FPGA for solving different kinds of problems. The efficiency of FPGA-based algorithms are higher than these CPU-based ones according to the experiments and simulation results.

Clearly, other intelligent optimization algorithms and operators can also be implemented in FPGA board to form a library as well as in general computer. The design and implementation of the configured SA based on FPGA described above is just a case for providing a reference. Its searching capacity and accuracy are to be improved further.

References

1. Carlier J, Pinson E (1989) An algorithm for solving the job-shop problem. Manage Sci 35(2):164–176
2. Lawler EL (1990) A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. Ann Oper Res 26(1):125–133
3. Lenstra JK, Kan AHGR, Brucker P (1977) Complexity of machine scheduling problems. Ann Discret Math 1:343–362
4. Blackstone JH, Phillips DT, Gary L (1982) A state-of-the-art survey of dispatching rules for manufacturing job shop operations. Int J Prod Res 20(1):27–45
5. Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Manage Sci 34(3):391–401
6. Hoitomt DJ, Luh PB, Pattipati KR (1993) A practical approach to job-shop scheduling problems. IEEE Trans Robot Autom 9(1):1–13
7. Rego C, Duarte R (2009) A filter-and-fan approach to the job shop scheduling problem. Eur J Oper Res 194(3):650–662
8. Gonçalves JF, de Magalhães Mendes JJ, Resende MCG (2005) A hybrid genetic algorithm for the job shop scheduling problem. Eur J Oper Res 167(1):77–95

9. Zhou H, Cheung W, Leung LC (2009) Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. Eur J Oper Res 194(3):637–649
10. De Giovanni L, Pezzella F (2010) An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. Eur J Oper Res 200(2):395–408
11. Lin TL, Horng SJ, Kao TW, Chen YH, Run RS, Chen RJ, Kuo I (2010) An efficient job-shop scheduling algorithm based on particle swarm optimization. Expert Syst Appl 37(3):2629–2636
12. Kasemset C, Kachitvichyanukul V (2012) A PSO–based procedure for a bi–level multi–objective TOC–based job–shop scheduling problem. Int J Oper Res 14(1):50–69
13. Zhang R, Wu C (2010) A hybrid immune simulated annealing algorithm for the job shop scheduling problem. Appl Soft Comput 10(1):79–89
14. Song SZ, Ren JJ, Fan JX (2012) Improved simulated annealing algorithm used for job shop scheduling problems. Advances in electrical engineering and automation. Springer, Berlin, pp 17–25
15. Yuan Y, Xu H (2013) Flexible job shop scheduling using hybrid differential evolution algorithms. Comput Ind Eng 65(2):246–260
16. Wisittipanich W, Kachitvichyanukul V (2012) Two enhanced differential evolution algorithms for job shop scheduling problems. Int J Prod Res 50(10):2757–2773
17. Burdett RL, Kozan E (2010) A disjunctive graph model and framework for constructing new train schedules. Eur J Oper Res 200(1):85–98
18. Bierwirth C, Mattfeld DC, Kopfer H (1996) On permutation representations for scheduling problems. Parallel problem solving from nature—PPSN IV, Springer, Berlin, pp 310–318
19. Giffler B, Thompson GL (1960) Algorithms for solving production-scheduling problems. Oper Res 8(4):487–503
20. Ponsich A, Tapia MGC, Coello CAC (2009) Solving permutation problems with differential evolution: an application to the jobshop scheduling problem. In: Proceedings of the 9th IEEE international conference on intelligent systems design and applications, pp 25–30
21. Golmakani HR, Namazi A (2012) An artificial immune algorithm for multiple-route job shop scheduling problem. Int J Adv Manuf Technol 63(1–4):77–86

Part VI
Future Works of Configurable Intelligent Optimization Algorithm

Chapter 12
Future Trends and Challenges

In this chapter, we give some future trends and challenges of dynamic configuration not only for intelligent optimization algorithm, but also for other algorithms used in the whole life cycle of manufacturing. Firstly, some works related to configuration of intelligent optimization algorithm are introduced. They have similar idea and can be further developed with different kinds of configuration ways. From the perspective of software improvement, we introduce the way of dynamic configuration for other algorithms in manufacturing. From the perspective of hardware improvement, the further development of dynamic configuration on FPGA for lightweight optimization in design, production and maintenance of manufacturing is given. Based on these trends, some challenges in developing dynamic configuration from different angles are listed in this chapter.

12.1 Related Works for Configuration of Intelligent Optimization Algorithm

Intelligent optimization algorithm, also named as meta-heuristic, has been developed for years. Hundreds of evolutionary schemes are presented with population-based iteration and operators. For improving its problem solving capability and make full use of the existing operators, many hybrid mechanisms are emerged. Two of the most famous mechanisms are hyper-heuristic and multi-method search. Hyper-heuristic [1, 2] mainly refers to a heuristic which tries to obtain right methods from a bunch of heuristics for solving a specific problem efficiently. The selection scheme itself, in hyper-heuristic, can be a kind of machine learning techniques or an adapting or turning process. Multi-method search [3, 4] then means to run multiple optimization algorithms simultaneously with population division and combination. The searching process is similar with the parallel configuration of algorithm hybridization mentioned in Chap. 4.

F. Tao et al., *Configurable Intelligent Optimization Algorithm*,
Springer Series in Advanced Manufacturing,
DOI: 10.1007/978-3-319-08840-2_12

(1) Hyper-heuristic

In hyper-heuristic, the goal is to select right algorithms for a specific problem. It is the main difference between hyper-heuristic and meta-heuristic. The selection is based on a bunch of existing algorithms and their performance to some degree. The method for selection is called high-level heuristic, while the algorithms solving the problem in different steps are called low-level heuristics. From the selecting process point of view, it is similar with our dynamic configuration way. More general than dynamic configuration in intelligent optimization algorithm, it tries to choose or combine several kinds of heuristics and machine learning techniques to solve a problem with specific framework [5] step by step. That is to say, it manages a number of heuristics and applies them to different stages of problem-solving. But its basic premise is that the framework or the process for solving a problem is clearly known. It is quite problem-dependent.

Hyper-heuristic is widely studied and applied in different area. With its development, Burke et al. [6, 16] have summarized its main classifications from structure to function. It consists of heuristic selection and heuristic generation, which are similar with our configuration methods in hybridation and in improvement respectively. The main focuses are the design of high-level heuristic based on a known framework, such as [7–9]. In manufacturing, it just applied for combinatorial optimizations such as production scheduling [10, 11], assembly line sequencing [12] and so on. For numerical optimization, parameter optimization and detection problems in manufacturing, few researches have been carried out. Also, the construction of hyper-heuristics largely depends on existing intelligent optimization algorithms [13, 14], as well as our configuration ways.

Broadly, the dynamic configuration ways in intelligent optimization algorithm which encapsulates operators as modules can be seen as a kind of hyper heuristics. That is because it also combines some low level operators (as heuristics) to solve different problems in iteration with some rules. But with generation division especially in intelligent optimization algorithm, dynamic configuration contains not only low level combination of operators, but also high level combination of algorithms. More than that, configuration has different types of schemes for algorithm improvement, hybridation and parallelization. It is independent from problems. Therefore, dynamic configuration is different with hyper-heuristics. It is an extension of component design in intelligent optimization algorithm. All ‘disposable’ and ‘reusable’ [15, 16] operators and algorithms can be reused according to dynamic configuration ways. To some extent, it can also be seen as an extension of hyper-heuristics.

(2) Multi-method Search

Multi-method search is presented by Vrugt et al. [3] who try to overcome the 'no free lunch' theory by applying several algorithms simultaneously to a class of problems. The idea is as well as our configuration method in improvement and hybridation of intelligent optimization algorithm with a simpler style. It is also established based on the population-based iteration rule and some existing algorithms. Different with hyper-heuristics, it sees the operators of an algorithm as an entity and invokes different entities in different sub-population serially. Just as mentioned in the above chapters, if there's one algorithm suitable for the specific problem, it can lead others to searching with a right direction. Now it has been applied for solving both combinatorial and numerical optimization [4, 17].

Nowadays, multi-method search, also known as A Multialgorithm Genetically Adaptive Method (AMALGAM), has been used for optimization in soil and water assessment [18], stochastic inversion in aquifer structure identification [19] and inverse parameter estimation in coupled simulation of surface runoff and soil water flow [20] and so on. But it has not been applied in manufacturing. Moreover, no work has been carried out especially for the design and selection of algorithms during iteration. Therefore, the multi-method search also has long way to go.

In the area of intelligent optimization algorithm, it can be seen that the dynamic configuration contains more ways to reuse existing algorithms fully and widely than the above two mechanisms. Furthermore, just like hyper-heuristic, the idea of configuration can be extensively applied for other algorithms and implemented in other hardware. Here we will take some typical numerical algorithm as an example to show the dynamic configuration ways for other algorithms in manufacturing. Moreover, further dynamic configuration on FPGA will be elaborated as our future work. As a whole, some challenges on the further development of dynamic configuration in the area of manufacturing are given in this chapter.

12.2 Dynamic Configuration for Other Algorithms

Besides optimization, there are a lot of large-scaled complex numerical computing requirements existing not only in manufacturing part design, but also in system and process evaluation, such as large-scaled matrix operations, linear or nonlinear functions and ordinary or partial differential equations. All these calculation process can be completed by a bunch of basic numerical algorithms according to the specific precision demands. Different with intelligent optimization algorithm, these numerical algorithms have different structure and heavily rely on the specific problem. Therefore, the previously mentioned configuration in a uniform iterative process is not adaptable for other numerical algorithms.

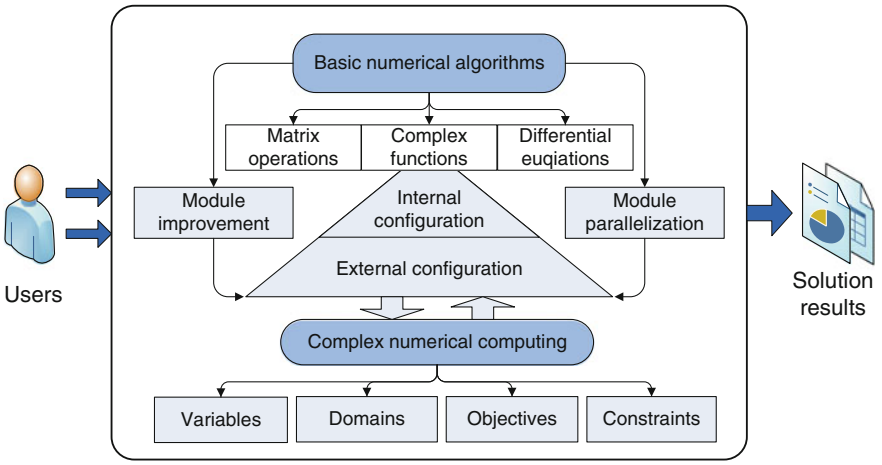


Fig. 12.1 The configuration framework for numerical computing

However, many complex numerical computing modules can be divided into several simple parts with some basic operation. And most numerical problems can be represented as a uniform form with complex variables and constraints. In such standardized form, these classical numerical algorithms can be directly invoked to solve the problem in different stages. Hence the numerical process is generally divisible and can be flexibly configured as well as intelligent optimization algorithm.

Considering three kinds of numerical computing, i.e. matrix operations, complex linear/nonlinear equations and ordinary/partial differential equations, existed generally in design, control and simulation of industrial manufacturing, the configuration framework can be drawn as Fig. 12.1. In this framework, we try to encapsulate basic numerical algorithms as modules and make two level configuration for different kinds of problems, i.e., (1) internal configuration and (2) external configuration.

As mentioned before, complex numerical computing can be divided into several steps. Each step that contains one or more standard numerical computing can also be represented by variables, domains, objectives and constraints. For example, if the numerical computing is linear equations, then it can be represented by a matrix $\mathbf{A}$ and a vector $\mathbf{b}$ (i.e. $\mathbf{Ax} = \mathbf{b}$). If the step needs to solve a partial differential equation, then it can be represented by a partial differential coefficient matrix from which a standard finite difference can be done based on that. In this case, the boundary conditions can be seen as constraints. With such standard representations in each computing stage, basic numerical algorithms which have been encapsulated as module can be configured and connected together.

In configuration, the internal configuration means to change the inner parameter of the module or to invoke other modules internally. For instance, there are many kinds of difference schemes for a specific partial differential equation. With a

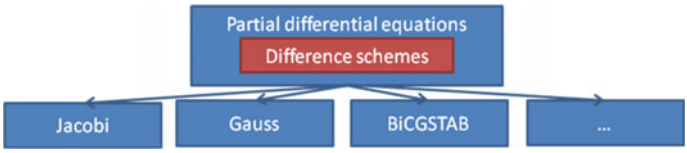


Fig. 12.2 Internal configuration for solving partial differential equations

difference scheme, partial differential equation can be transformed into a group of sparse linear equations. Then Jacobi or Gauss-Seidel method can be used to solve the linear equations. In this case, Jacobi method which is encapsulated into module can be configured with a suitable difference scheme to solve the numerical problem, as shown in Fig. 12.2. Making the transformation of difference scheme also as a module, the configuration above is called internal configuration. By means of multiple basic numerical modules, we could use some heuristic to intelligently select one or more of them connected together for solving a specific numerical problem. Overall, internal configuration in different kinds of numerical computing requires fine grained division of calculating modules. Moreover, all parameters should be adjustable with a standard form to make sure the modules can be correctly invoked.

External configuration, different with the internal one, refers to configure different algorithms in different solving stages for a complex problem. The configured module mainly refers to coarse-grained numerical algorithms. These algorithms can be either a single basic module such like matrix operation, or a combined module configured by the internal configuration as mentioned before. Take the design of aircraft wing for example, the process consists not only the key size design but also the verification of its aerodynamic characteristics. With a bunch of datum, matrix operation may be the first step, the solving of partial differential equations and size decision may be the next. On this occasion, we could configure matrix operation module, Gauss-Seidel and a kind of difference scheme and an intelligent optimization algorithm step by step for solving the whole problem. The outputs of the previous steps are the inputs of the latter ones. Together with internal configuration and external configuration, an integrated numerical algorithm can be generated by a specific heuristic (or algorithm selection rule) and a mass of basic numerical modules.

Moreover, as shown in Fig. 12.1, we could also do module improvement and parallelization during the whole process. Improvement can be easily obtained through flexible parameter interface as mentioned before. Parallelization here consists of both internal and external parallelization.

Internal parallelization means to directly encapsulate basic parallel numerical algorithms as modules. By way of configuring the parallel modules, some new parallel algorithms can be easily produced. As the same as in intelligent optimization algorithm, external parallelization refers to execute different modules simultaneously in different computing nodes. External parallelization is highly dependent on the computing process of a specific problem. For example, we could

separate irrelevant computing module manually to parallel nodes for high time efficiency. We also could apply different numerical algorithms for a single computing stage simultaneously in different computing nodes for high accuracy and stability.

As we know, the establishment of basic numerical modules has been carried out for years. There are already many serial and parallel tools for basic numerical computing, such as matrix operations and solver of linear equations. Basic linear algebra subprograms (BLAS) [21] and Parallel BLAS (PBLAS) [22], LINPACK [23] and PETSc [24] and so on are all famous and have been widely used in different areas. These tools based on different platform and programming language can not be integrated together. If we need to invoke these basic modules, we have to program our problem with the specific programming language and do more changes. With different formation, different numerical modules can not be connected and configured directly. Moreover, there has no recommendation to decide which is the most suitable one for a specific problem.

Therefore, for establishing a configurable platform for wider numerical computing, the uniform encapsulation of existing numerical algorithms is required. In other words, the interface of each module to be configured should be uniformed and full information should be provided for flexible configuration. After that, the most important thing of numerical configuration is the construction of rule base which can be used for algorithm selection according to the problem characteristics and the calculating environments.

12.3 Dynamic Configuration on FPGA

As elaborated in Chaps. 5 and 11, the dynamic configuration of not only intelligent optimization algorithm but also other numerical algorithm can be implemented on FPGA. Focus on the configuration of intelligent optimization algorithm, there are two implementing schemes on FPGA, as shown in Figs. 12.3 and 12.4, (1) operator-based configuration and (2) algorithm-based configuration. It should be noted that an FPGA board can only store limited operators for just one class of problems.

In the first scheme, we could firstly extract initialization part and population updating part into two modules. Operators in iteration can be implemented independently with uniform population-based interfaces. As shown in Fig. 12.3, the connection of these modules can generate a complete intelligent optimization algorithm. *Op1*, *Op2*, *Opx*, *Opy*, *Opz* and *Opn* represent different kinds of operators, such as single-point crossover and mutation in genetic algorithm. The red solid line and the blue chain dotted line represent two kinds of connection ways respectively. With only one connection path between initialization and population updating, a hybrid intelligent optimization algorithm can be generated in which all population is concurrently operated by the corresponding operators on FPGA. In contrast, if there are two or more connection paths, then the population will be

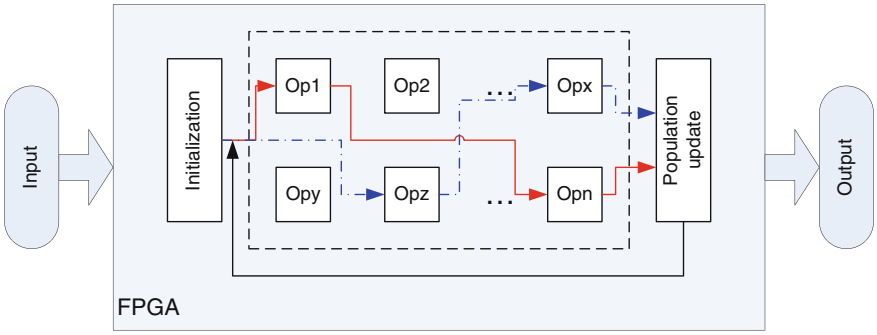


Fig. 12.3 Operator-based configuration on FPGA

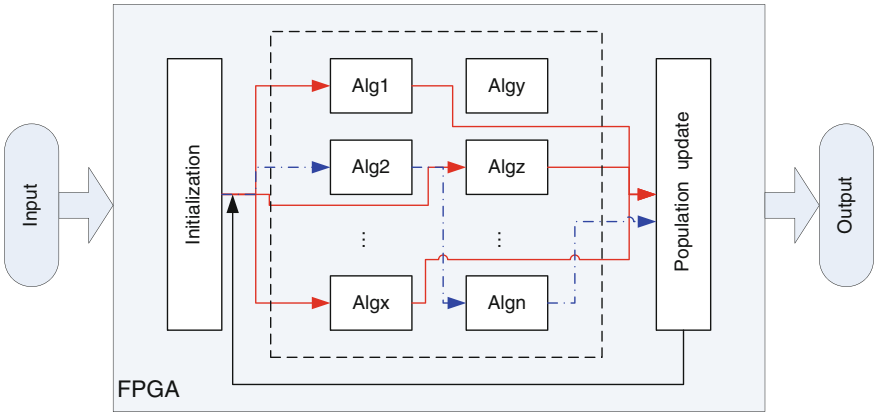


Fig. 12.4 Algorithm-based configuration on FPGA

evenly separated as multiple groups. Each sub-population is executed following the relevant path. A parallel intelligent optimization algorithm with several groups of hybrid operators can be produced. It is the same as the configuration in general cluster environment.

In the second scheme, we could implement the whole operators of a specific algorithm together as one module. For example, if *Alg1* is the classical GA, then the operations in it consist of population selection, crossover and mutation. As shown in Fig. 12.4, we could arbitrarily select different algorithm in parallel with multiple connection paths to form hybrid parallel intelligent optimization algorithm on FPGA. And we could also connect algorithms in serial to generate a single hybrid one. No matter which kinds of connection ways, parallel execution of individuals is ensured on FPGA with high time efficiency.

The implementation of intelligent optimization algorithm on FPGA mainly aims at lightweight and high speed decision in a special environments or devices. One implementation with fixed architecture is just for one specific problem and

can not be reused to others. With the above dynamic configuration, an FPGA board with several operators can then be widely applied to a large amount of problems without reconstruction. Although the implementation is under construction, we believe it will be practical and valuable for many problems ranging from production evaluation to maintenance to realize lightweight high efficiency decision and optimization.

12.4 The Challenges on the Development of Dynamic Configuration

In this book, we elaborated all kinds of dynamic configuration ways for improvement, hybridation and parallelization of intelligent optimization algorithm and their application in manufacturing field. On the basis of various modules, much more efficient algorithms can be generated for different complex problems ranging from part design to system management. A bran-new design conception is introduced for intelligent optimization algorithm. However, the new design method has shifted the difficulty from 'algorithm design' to 'algorithm selection'. Currently, the most important things to establish such dynamic configuration are the construction of algorithm module library and the establishment of recommend rules. The former task has been carried out, as mentioned in Chap. 3, while the latter one as a core part to realize intelligent dynamic configuration is still a big challenge.

Specifically, during the configuration process, designer or engineer have to know the existing operators well enough to select suitable ones for a specific problem. Balance between exploration and exploitation is highly required in configuration. However, to most engineers, it is still not easy to make decision. With less knowledge on different kinds of intelligent optimization algorithms, they have to do large amount of experiments to traverse all these operators and select one or two of them according to the results. It is time consuming. Hence, a rule base is required to provide recommendation in deciding which configuration way is suitable, which operators to select and how to set the parameters in the specific algorithms.

For establishing a rule base, we need to classify the existing problem into different kinds firstly. As introduced in Chap. 2, in the whole life cycle, large amount of numerical function optimization, parameter optimization, detection and classification, combinatorial scheduling and multi-disciplinary optimization exist in not only product design, but also process and system management. However, such classification is far from enough. Each category can further be divided into two kinds in accordance with whether the variables are continuous or discrete. Different variables are represented with different encoding scheme. So that one operator in different encoding scheme is totally different. According to its variables, the problem can be classified as continuous problem, non-sequencing discrete problem

and sequencing discrete problem. For example, traveling salesman problem belongs to sequencing discrete problem, in which its variables are a serial integral number and are different from each other, while traditional task scheduling problem is a kind of non-sequencing discrete problem. Of course in many situations the variables are a blend of continuous and discrete ones. With such classification, when a problem is modeled and submitted, a rule must identify which category it belongs to.

Based on the classification, the second step is to classify the existing operators according to the variable characteristics. For example, the operator of classical ant colony algorithm is suitable especially for sequencing discrete problem, while the operator of traditional particle swarm optimization belongs to continuous operation. With clear operations, this step is easy to implement.

After that, a bunch of heuristics or some learning algorithms, as well as hyper-heuristics, are required for further determining how to select a group of operators and configure them for a specific problem. This is crucial and hard to implement.

On one hand, theoretical verifications of intelligent optimization algorithms are quite less. For a class of problems, we need to take large number of experiments to see whether an operator is suitable. That is very time consuming. If the problem is changed, more tests have to taken to adjust the variation. It can be seen that, the relation between an operator and a class of problems is hard to figure out.

On the other hand, if we have obtained a group of operators, the problems of which configuration approach to use and how to configure them together are also two difficulties for us. From a learning point of view, the construction of recommendation rule also requires large amount of experimental and practical data from a real system.

For overcoming these challenges and establishing a certain level of automatic configuration, a lot of data obtained from a number of experiments based on different sorts of problems and the design of high level machine learning algorithms are both imperative.

In spite of this, dynamic configuration of intelligent optimization algorithm can still be applied with manual control in various kinds of problems not only in manufacturing but also in other fields. Especially by means of parameter-based configuration, operator-based configuration and algorithm-based configuration in both serial and parallel algorithm design, limit operators can produce hundreds of algorithms in different platforms. As a new design mechanism, it can solve wider dynamic problems with uncertainties and complex components through diverse configurations and a bunch of tests.

12.5 Summary

In this chapter, we mainly talked about some related works similar with the dynamic configuration of intelligent optimization algorithm. The similarities and differences between the dynamic configuration and hyper-heuristics and multi-method search are given. Moreover, we showed that the idea of dynamic

configuration can be further developed for other algorithms and on different kinds of hardware platforms. Some future design framework of dynamic configuration for numerical algorithms in manufacturing is drawn. And two kinds of flexible implementations of dynamic configurable intelligent optimization algorithms on FPGA are elaborated.

Currently, all these works are under construction. Each of them can be applied in different area for fast and efficient design and optimization. Further, for realizing automatic configuration, as well as in hyper-heuristics, we summarized some challenges on the development of rule base for dynamic configuration. It is one of the most crucial components which is imperative for engineers and designers, who do not have the comprehensive knowledge of intelligent optimization algorithm, to apply such configuration schemes based on a configurable intelligent optimization algorithm library.

References

1. Burke EK, Hart E, Kendall G, Newall J, Ross P, Schulenburg S (2003) Hyper-heuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer Academic Publishers, Boston
2. Ross P (2005) Hyper-heuristics, search methodologies. Springer, Berlin
3. Vrugt JA, Robinson BA (2007) Improved evolutionary optimization from genetically adaptive multimethod search. Proc Natl Acad Sci 104(3):708–711
4. Vrugt JA, Robinson BA, Hyman JM (2009) Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Trans Evol Comput 13(2):243–259
5. Qu R, Burke EK (2009) Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. J Oper Res Soc 60:1273–1285
6. Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. Springer, New York, pp 449–468
7. Hart E, Ross P, Nelson JAD (1998) Solving a real-world problem using an evolving heuristically driven schedule builder. Evol Comput 6(1):61–80
8. Ochoa G, Qu R, Burke EK (2009) Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In: Proceedings of the ACM genetic and evolutionary computation conference (GECCO), pp 341–348
9. Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph-based hyper-heuristic for educational timetabling problems. Eur J Oper Res 176:177–192
10. Vazquez-Rodriguez JA, Petrovic S, Salhi A (2007) A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines. In: Proceedings of the 3rd multidisciplinary international scheduling conference: theory and applications (MISTA)
11. Rodríguez JAV, Salhi A (2007) A Robust meta-hyper-heuristic approach to hybrid flow-shop scheduling. In: Evolutionary scheduling. Springer, Berlin, pp 125–142
12. Cano-Belmán J, Ríos-Mercado RZ, Bautista J (2010) A scatter search based hyper-heuristic for sequencing a mixed-model assembly line. J Heuristics 16(6):749–770
13. Bai R, Burke EK, Kendall G (2007) Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. J Oper Res Soc 59(10):1387–1397

14. Burke EK, Kendall G, Soubeiga E (2003) A tabu-search hyperheuristic for timetabling and rostering. J Heuristics 9(6):451–470
15. Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward J (2009) Exploring hyper-heuristic methodologies with genetic programming. In: Mumford C, Jain L (eds) Collaborative computational intelligence. Springer, Berlin, pp 177–201
16. Burke EK, Hyde M, Kendall G, Woodward J (2010) A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. IEEE Trans Evol Comput 14(6):942–958
17. Vrugt JA, Robinson BA, Hyman J (2008) A universal multimethod search strategy for computationally efficient global optimization. Geological Society of America (GSA), New York, pp 28–31
18. Zhang X, Srinivasan R, Liew MV (2010) On the use of multi-algorithm, genetically adaptive multi-objective method for multi-site calibration of the SWAT model. Hydrol Process 24(8):955–969
19. Harp DR, Dai Z, Wolfsberg AV, Vrugt JA, Robinson BA, Vesselinov VV (2008) Aquifer structure identification using stochastic inversion. Geophys Res Lett 35(8):L08404
20. Köhne JM, Wöhling T, Pot V, Benoit P, Leguédois S, Bissonnais YL, Šimůnek J (2011) Coupled simulation of surface runoff and soil water flow using multi-objective parameter estimation. J Hydrol 403(1):141–156
21. Lawson CL, Hanson RJ, Kincaid D, Krogh FT (1979) Basic linear algebra subprograms for FORTRAN usage. ACM Trans Math Softw 5:308–323 (Algorithm 539)
22. Choi J, Dongarra J, Ostrouchov S, Petitet A, Walker D, Whaley RC (1996) A proposal for a set of parallel basic linear algebra subprograms. In: Applied parallel computing computations in physics, chemistry and engineering science. Springer, Berlin, pp 107–114
23. Dongarra JJ (ed) (1979) LINPACK users' guide, vol 8. Siam, Philadelphia
24. Balay S, Gropp WD, McInnes LC, Smith BF (1996) PETSc 2.0 users manual. Mathematics and computer science division (UC-405), Argonne National Laboratory