

# 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

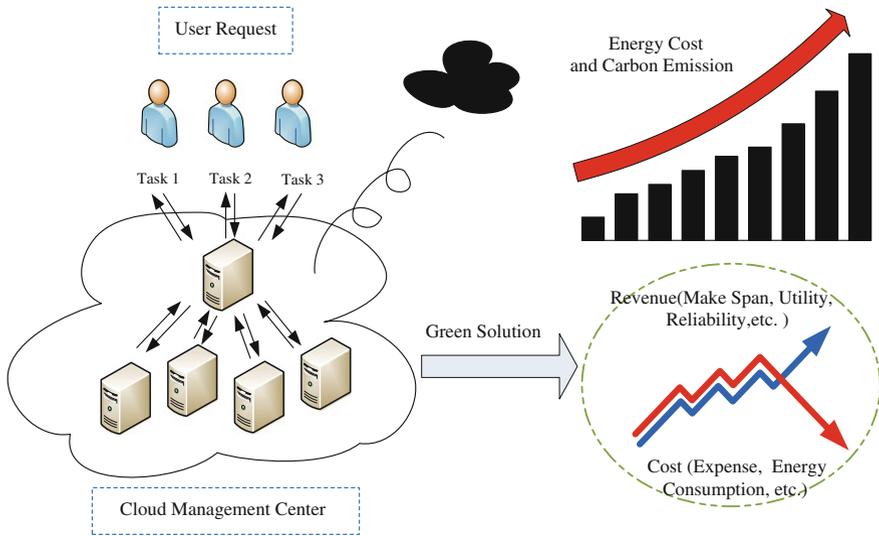
Cloud computing is a new technique based on distributed computing, parallel computing and grid computing. Numerous implementation plans from various major software enterprises have been proposed, including the Amazon Web Services (AWS, Amazon Web services), Google's App Engine cloud computing platform, 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.

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 incorporated 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 includes 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  $T_i \in \{1, \dots, T_{\max}\}$  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.

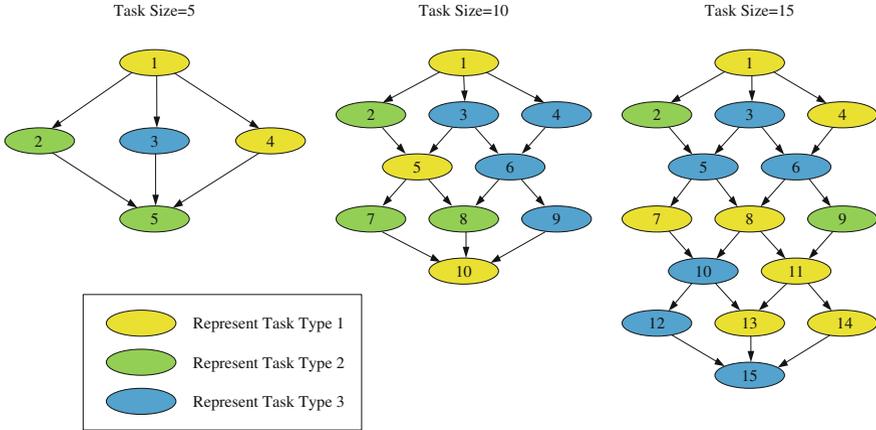


Fig. 7.2 DAGs of user requests

**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  $\mathbf{TP}$ .
- $\mathbf{S}$ : 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.
- $\mathbf{DC}$ : denotes the bandwidth between processors, where  $DC_{kl}$  represents the transferring rate of data from processor  $P_k$  to processor  $P_l$ .
- $\mathbf{EC}$ : 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  $\forall i \in \{1, 2, \dots, n\}$ ,  $X(i) \in \{1, 2, \dots, p\}$ .

### 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

$$TComplete(i) = \max_{j=1:i-1} \left\{ E_{ji} \times TComplete(j) + \frac{E_{ji} \times DOUT_j}{DC_{X(j)X(i)}} \right\} + TP_{T_i X(i)} DIN_i \quad (7.1)$$

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:

$$LoadBalance = \sum_{k=1}^p (LoadPortion(k) - BestPortion(k))^2 \quad (7.2)$$

where,

$$LoadPortion(k) = \frac{\sum_{i=1}^n DIN_i |X(i) = k}{\sum_{i=1}^n DIN_i} \quad (7.3)$$

$$BestPortion(k) = \frac{S_k / \overline{TP}_k}{\sum_{k=1}^p S_k / \overline{TP}_k} \quad (7.4)$$

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:

$$FinalTComplete = TComplete(n) \times e^{\alpha \cdot LoadBalance} \quad (7.5)$$

In Eq. (7.5),  $\alpha$  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  $\alpha$  is set to represent the possible delay on makespan caused by imbalance in load distribution. While the network is idle,  $\alpha$  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.

$$EnergyConsumed = EnergyComp + EnergyComm \quad (7.6)$$

$$EnergyComp = \sum_{i=1}^n EP_{T_i X(i)} TP_{T_i X(i)} DIN_i \quad (7.7)$$

$$EnergyComm = \sum_{i=1}^n \sum_{j=1}^{i-1} \frac{E_{ji} DOUT_j}{DC_{X(j)X(i)} / EC_{X(j)X(i)}} \quad (7.8)$$

### 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:

$$Min \left( TComplete(n) \times e^{\alpha \sum_{k=1}^p \left( \frac{\sum_{i=1}^n DIN_i |X(i)=k}{\sum_{i=1}^n DIN_i} \frac{S_k / \overline{TP}_k}{\sum_{k=1}^p S_k / \overline{TP}_k} \right)^2} \right) \quad (7.9)$$

$$Min \left( \sum_{i=1}^n EP_{T_i X(i)} TP_{T_i X(i)} DIN_i + \sum_{i=1}^n \sum_{j=1}^{i-1} \frac{E_{ji} DOUT_j}{DC_{X(j)X(i)} / EC_{X(j)X(i)}} \right) \quad (7.10)$$

## (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:

$$X(i) \in \{1, 2, \dots, p\}, \forall i \in \{1, 2, \dots, n\} \quad (7.11)$$

$$TP_{T,X(i)}DIN_i \leq MaxTProcessing \quad (7.12)$$

$$\max_{j=0:i-1} \left\{ \frac{E_{ji} \times DOUT_j}{DC_{X(j)X(i)}} \right\} \leq MaxTCommunication \quad (7.13)$$

$$EP_{T,X(i)}TP_{T,X(i)}DIN_i \leq MaxTProcessing \quad (7.14)$$

$$\sum_{j=0}^{i-1} \frac{E_{ji}DOUT_j}{DC_{X(j)X(i)}/EC_{X(j)X(i)}} \leq MaxECommunication \quad (7.15)$$

$$\frac{\sum_{i=1}^n DIN_i |X(i) = k}{\sum_{i=1}^n DIN_i} \leq UpperLPortion(k) \quad (7.16)$$

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.

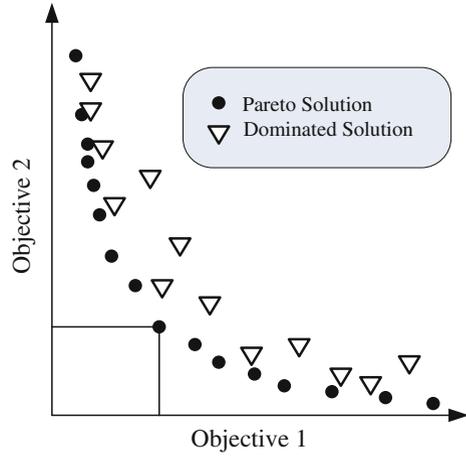
#### 7.4.1.1 Domination and Non-Inferiority

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.

$$Distance(i) = \sum_{j=1}^{NObj} Dis(i, j), \forall i = 1 : NInd \tag{7.17}$$

$$Dis(index(i), j) = \begin{cases} \min(\inf, data(index_j(i + 1), j) - data(index_j(i - 1), j)), & \forall i = 2 : NInd - 1 \\ \inf, & i = 1 \text{ or } i = NInd \end{cases} \tag{7.18}$$

$$Score(i, j) = \frac{score(i, j)}{1 + \max_{k=1:NInd, score(k, j) \neq \inf} (|score(k, j)|)}, \forall i = 1 : NInd, j = 1 : NObj \tag{7.19}$$

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<sub>j</sub>* 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  $\min\{ParetoFraction \times PopulationSize, Numbers\ Existing\ in\ the\ Pareto\ Front\}$ .

### 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.

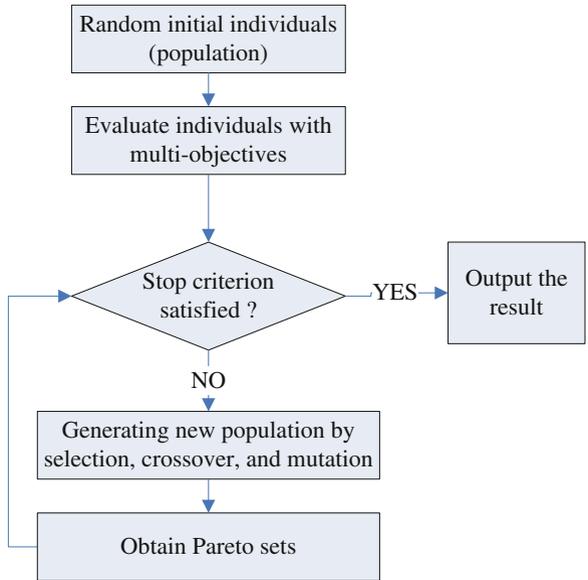


Fig. 7.4 Framework of GA in MOO

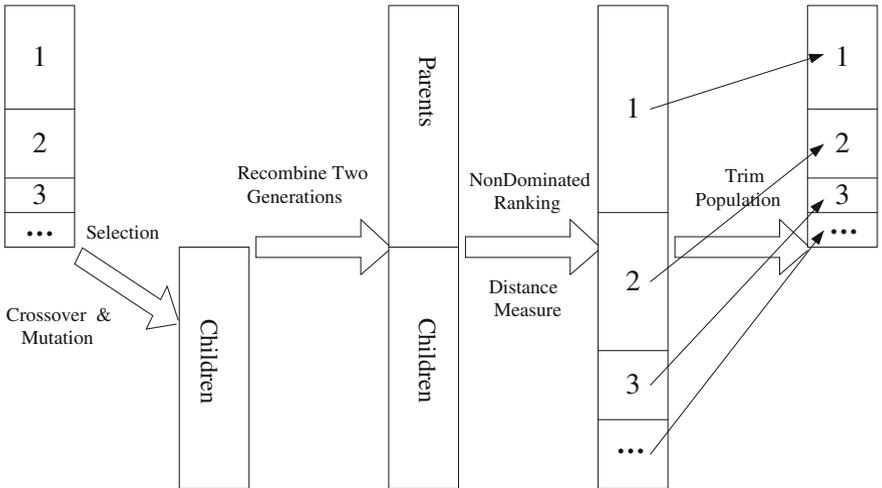


Fig. 7.5 Evolution process of GA in MOO

**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: $1e^{-3}$

$$SpreadChange(gen) = \begin{cases} \sum_{i=1}^{StallGen} \left( Weight^{StallGen+1-i} \times \frac{|Spread(i+1) - Spread(i)|}{1 + Spread(i)} \right), & gen > StallGen \\ \text{inf}, & \text{otherwise} \end{cases} \quad (7.20)$$

$$Spread = \frac{extremeParetoDistance + avgDistance}{extremeParetoDistance + NObj \times avgdistance} \quad (7.21)$$

$$extremeParetoDistance = \sum_{i=1}^{NObj} \|extremeParetoSold(i) - extremeParetoSNew(i)\| \quad (7.22)$$

$$avgdistance = \frac{\sum_{i=1}^{NInd} Distance(i)}{NInd} \quad (7.23)$$

$$avgdistance = \frac{\|Distance - avgDistance\|}{\sqrt{NInd}} \quad (7.24)$$

$$MeanSpread(gen) = \frac{\sum_{i=gen-StallGen}^{gen} Spread(i)}{StallGen + 1}, \quad gen > StallGen \quad (7.25)$$

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 problems by 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  $-0.5 \leq a_i \leq 0.5$ . 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:  $X^*(j) = \arg \max \{a(i, j) | i \in \{1, 2, 3, \dots, M\}\}$ , 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  $j$ th 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  $odds_a(i, j)$ .

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 cross-over 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  $\delta$ , i.e.  $S \geq \delta$ , where  $\delta$  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:

$$S = \frac{k_t}{n} \times \frac{k_e}{m} \quad (7.26)$$

where  $k_t$  represents the number of identical values in the task type vectors ( $\mathbf{T}$ ) 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 ( $\mathbf{E}$ ),  $n$  represents the number of task units, and  $m$  represents the number of elements with value of 1 in  $\mathbf{E}$  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  $\mathbf{T}$ ,  $\mathbf{E}$  and  $\mathbf{Tcase}$ ,  $\mathbf{Ecase}$  given below, is computed as follows:

$$\mathbf{T} = [1 \ 2 \ 2], \mathbf{Tcase} = [1 \ 3 \ 2], \mathbf{E} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{Ecase} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

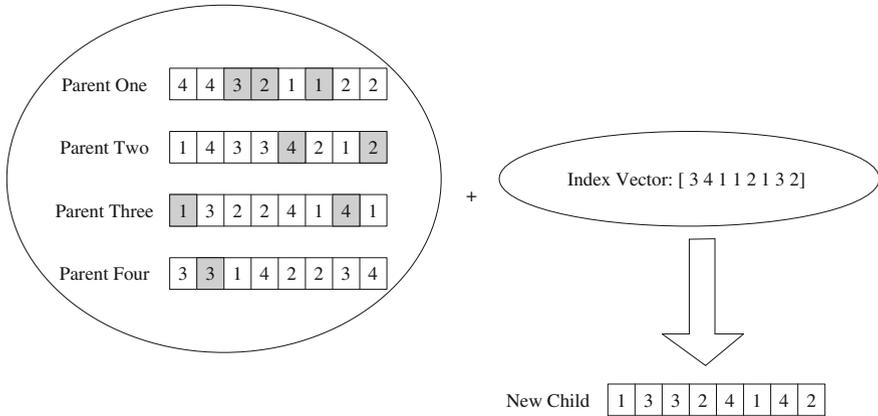
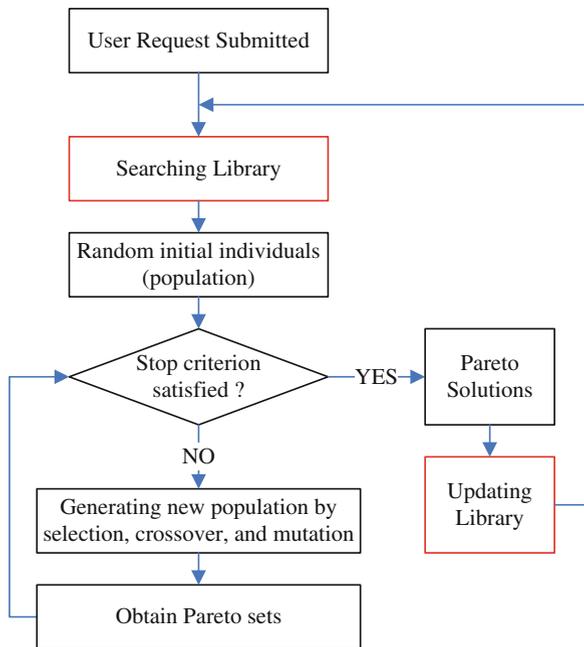


Fig. 7.6 Instance of MPCO

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 **case** = {**TCase**, **ECase**, **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 **case\_1** = {**TCase**, **ECase**, **Solutions**} is stored. Then **case\_1** will point to the second case labeled as **case\_2** = {**TCase**, **ECase**, **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:

$$C(i) = \frac{1}{N} \sum_{j \in N} Similar(i, j) \quad (7.27)$$

$$Similar(i, j) = \begin{cases} 1, & S(i, j) > \gamma \\ 0, & Otherwise \end{cases} \quad (7.28)$$

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).  $\gamma$  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  $\delta$  is set to be 0.8, then no similar cases exist, and hence the initial population shall be randomly generated. But if  $\delta$  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.

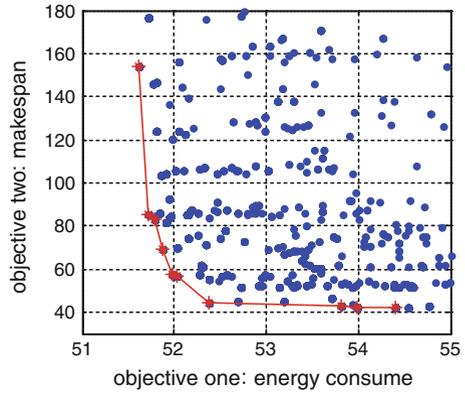
### 7.5.2 Experiments and Results

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  $\mathbf{T}$  and matrix  $\mathbf{TP}$ , 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 2 has 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	$8.16 \times 10^{-8}$	32.1	876.8
	MPCO	0.0012	$4.8 \times 10^{-8}$	30.7	652.4
Task number = 10	TPCO	0.0014	$2.92 \times 10^{-7}$	33.9	467.6
	MPCO	0.0014	$2.48 \times 10^{-7}$	35.3	448.9
Task number = 15	TPCO	0.0023	$1.13 \times 10^{-7}$	41.4	528.4
	MPCO	0.0026	$2.32 \times 10^{-7}$	42.8	456.7
Task number = 20	TPCO	0.0026	$7.27 \times 10^{-7}$	57.7	572.7
	MPCO	0.0033	$4.14 \times 10^{-6}$	56.5	648.9
Task Number = 30	TPCO	0.0029	$4.38 \times 10^{-7}$	83.4	1217.6
	MPCO	0.0036	$1.43 \times 10^{-6}$	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  $1.43 \times 10^{-6}$  for MPCO versus 0.0029 and  $4.38 \times 10^{-7}$  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.

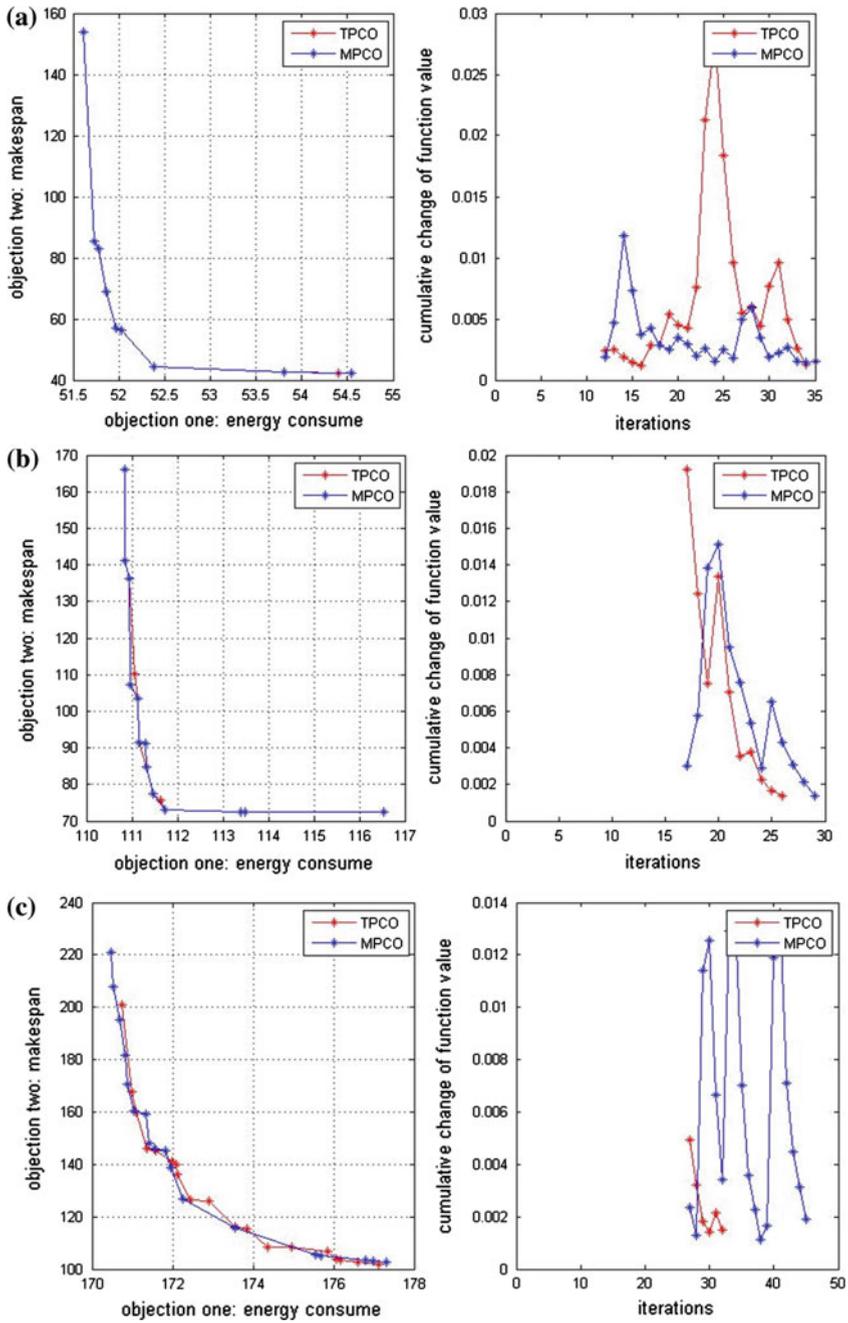


Fig. 7.9 Comparison of TPCO and MPCO as task size increases from 5 (top) to 30 (bottom)

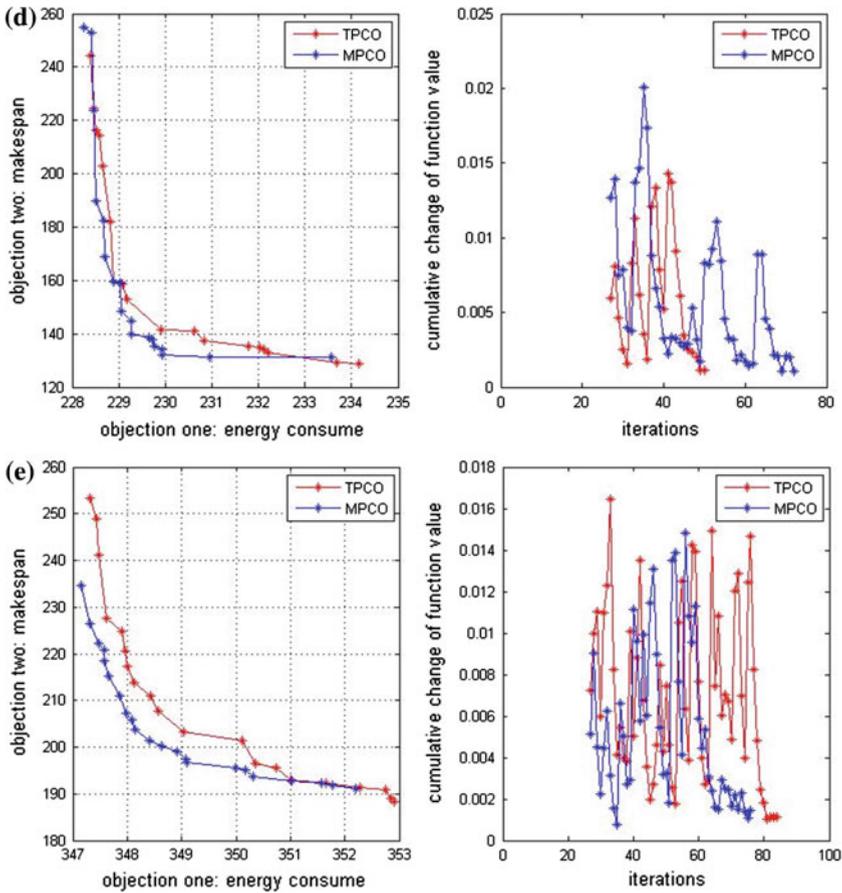


Fig. 7.9 continued

### 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.

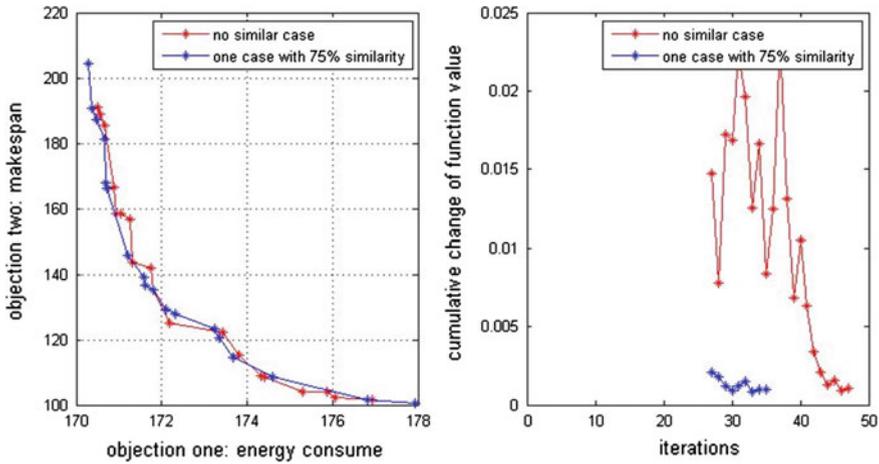
In summary, the practice of introducing cases of high similarity to help with initialization can effectively speed up the convergence rate and improve the evolution stability of the algorithm. The appropriate similarity threshold can be set as 75 %. It should also be noted that the evolution and convergence 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	$7.29 \times 10^{-4}$	$6.97 \times 10^{-4}$	$7.42 \times 10^{-4}$	$7.38 \times 10^{-4}$	$7.05 \times 10^{-4}$
Mean of SpreadChange variance	$4.82 \times 10^{-5}$	$5.13 \times 10^{-5}$	$3.09 \times 10^{-5}$	$2.61 \times 10^{-5}$	$1.88 \times 10^{-5}$



**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 *SpeedChange* 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 <i>SpreadChange</i> in last generation	Mean of <i>SpreadChange</i> variance
AGA	77.6	501.5	0.0018	$3.82 \times 10^{-7}$	$6.41 \times 10^{-4}$	$3.56 \times 10^{-5}$
CGA	102.5	1879.9	0.0022	$1.96 \times 10^{-7}$	$5.25 \times 10^{-4}$	$7.44 \times 10^{-5}$
LGA	43.7	1035.7	0.0016	$2.08 \times 10^{-7}$	$6.78 \times 10^{-4}$	$6.31 \times 10^{-5}$
<b>CLPS-GA</b>	36.4	355.5	0.0020	$4.94 \times 10^{-8}$	$7.38 \times 10^{-4}$	$2.61 \times 10^{-5}$

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  $2.61 \times 10^{-5}$ , 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

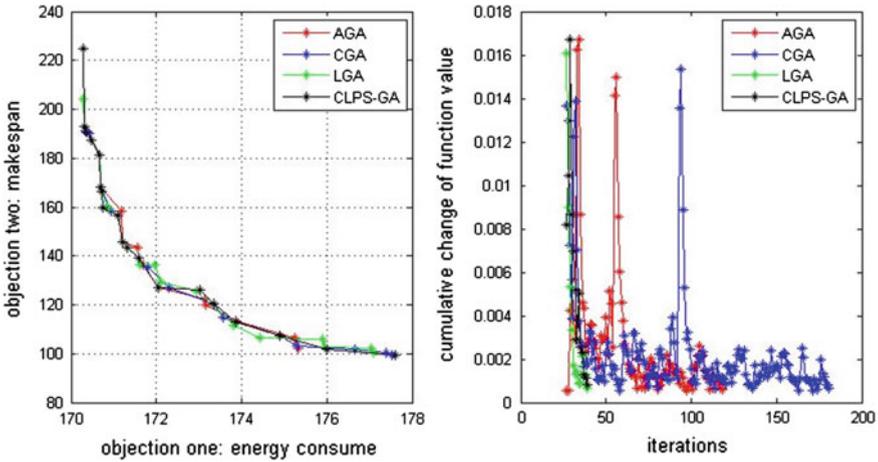


Fig. 7.11 Comparison of CLPS-GA and other enhanced GAs

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, Griffith 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, Beijing, 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, Australia, 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. Vafae 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. Wiley, 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