Intelligent Computational Optimization in Engineering: Techniques and Applications

Lars Nolle, Mario Köppen, Gerald Schaefer, and Ajith Abraham

Abstract. Many problems can be formulated as optimization problems. Among the many classes of algorithms for solving such problems, one interesting, biologically inspired group is that of meta-heuristic optimization techniques. In this introductionary chapter we provide an overview of such techniques, in particular of Genetic Algorithms, Ant Colony Optimization and Particle Swarm Optimization techniques.

1 Introduction

Many engineering problems involve heuristic search and optimization where, for example, an input parameter vector for a given system has to be found in order to optimize the system response. Also, many engineering optimization problems, once discretized, may become combinatorial in nature, which gives rise to certain difficulties in terms of solution procedure. In the first instance many problems have enormous search spaces, are NP-hard and hence require heuristic solution techniques. A second difficulty is the lack of ability of classical solution techniques to determine appropriate (global) optima of non-convex problems involving numerous (local) optima. Under these conditions, approaches based on recent advances

Lars Nolle
School of Science and Technology, Nottingham Trent University, U.K.
e-mail: lars.nolle@ntu.ac.uk

Mario Köppen
NDRC, Kyushu Institute of Technology
e-mail: mkoeppen@ieee.org

Gerald Schaefer
Department of Computer Science, Loughborough University
e-mail: G.Schaefer@lboro.ac.uk

Ajith Abraham
Machine Intelligence Research Labs (MIR Labs)
e-mail: ajith.abraham@ieee.org

M. Köppen et al. (Eds.): Intelligent Comput. Optimization in Eng., SCI 366, pp. 1–24.
springerlink.com © Springer-Verlag Berlin Heidelberg 2011

in computational optimization techniques have been shown to be advantageous and successful compared to classical approaches.

The single contributions of this book detail the successful use of computational optimization techniques that use (meta-)heuristic search to solve non-convex and complex engineering problems. In the following, we want to give a short overview of basic meta-heuristic optimization techniques from a more academic perspective, before detailing the application aspects of these methods.

2 Evolutionary Computing

Many scientific problems can be viewed as search or optimization problems, where an optimum input parameter vector for a given system has to be found in order to maximise or to minimise the system response to that input vector. Often, auxiliary information about the system, like its transfer function and derivatives, is not known and the measures might be incomplete and distorted by noise. This makes such problems difficult to be solved by traditional mathematical methods. Here, evolutionary optimization algorithms, which are based on biological principles borrowed from nature, can offer a solution. These algorithms work on a population of candidate solutions, which are iteratively improved so that an optimal solution evolves over time.

This chapter discusses the general problem of search and optimization before it introduces the systems view, followed by a definition of search space and fitness landscape. It then explains the process of optimization and the concept of optimization loops. It continuous with introducing biological-inspired evolutionary optimization algorithms, namely Genetic Algorithms and Genetic Programming. Other evolutionary inspired optimization techniques, namely Ant Colony Optimization and Particle Swarm Optimization are also discussed.

2.1 Systems

Every process or object can be seen as a system. Fenton and Hill [9] define a system as "*...an assembly of components, connected together in an organised way, and separated from its environment by a boundary. This organised assembly has an observable purpose which is characterised in terms of how it transforms input from the environment into output to the environment.*" By definition, a system has exactly one input channel x and exactly one output channel y (Figure 1). All interactions with the environment have to be made through these interfaces.

Both input and output can be vectors or scalars. The input is called the *independent variable* or *parameter*, because its value(s) can be chosen freely, and results in the output y, the so-called *dependent variable*. If the present state of the system does not depend on previous states but only on the current input, the system is said to be a *steady state system*, the output of the system can be described as a function of the input $y = f(x)$.

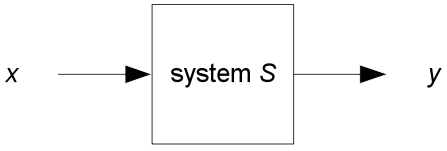

Fig. 1 Generic system.

2.2 Objective Function

In order to rate the quality of a candidate solution x, it is necessary to transform the system response to x into an appropriate measure, called the *objective* or *fitness*. If the system has only one output variable, the system output y equals the fitness. If y has more than one component the output variables of the system have to be combined into a single value, computed by the so called *objective function* or *fitness function*. In general, there are four approaches to judge the system output: *aggregation*, the *Changing Objectives Method*, the Use of *Niche Techniques* and *Pareto Based Methods* [10]. The most often used method is aggregation. In its simplest case, the fitness function $F(x)$ equals the weighted sum of the components $y_i = c_i \cdot F_i(x)$ of y, where c_i is the weight for component i:

$$F(x) = c_0 + c_1 \cdot F_1(x) + \cdots + c_n \cdot F_n(x) \tag{1}$$

2.3 Search Space and Fitness Landscape

If all the possible candidate solutions are collected in an ordered way, this collection is called the *search space*. Sometimes, this space is also referred to as input space. For an optimization problem of dimension n, i.e. a system with n independent parameters, the search space also has dimension n. By adding the dimension *Fitness* or *Costs* to the search space, one will get the $(n+1)$-dimensional *fitness landscape* [23].

2.4 Optimization

Optimization [21] is the process of selecting the best candidate solution from a range of possibilities, i.e. the search space. In other words, a system S, that has to be optimized in terms of a quality output value y, is brought into a new state that has a better quality output value y than the previous state. This is done by changing the independent input parameters x. The error function describes the difference between the predefined objective $y_{desired}$ and systems response $f(x)$ to the input x.

$$Error(x) = y_{desired} - f(x) \tag{2}$$

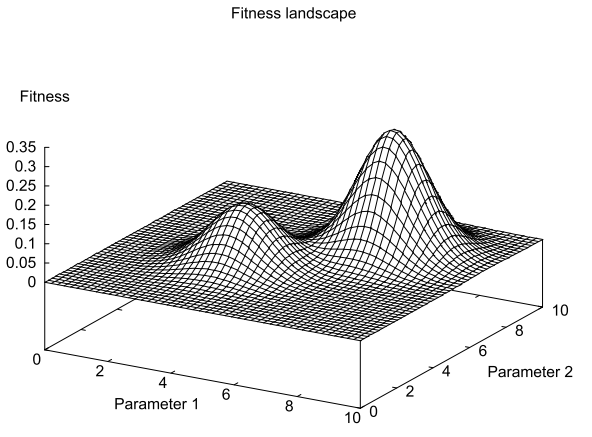

Fig. 2 Example of a fitness landscape for a system with two input parameters.

Usually, the aim is to find the vector x' that leads to a minimal error for the system S, i.e. the minimal departure from the optimal output value:

$$Error(x') = 0 \tag{3}$$

Often, a predefined target value is not known. In this case one tries to gain a fitness value that is as high as possible in case of maximisation, or as low a possible in the case of minimisation.

Ideally, one would evaluate all possible candidates and choose the best one. This is known as exhaustive search. However, often it is not feasible to consider all possible solutions, for example if the search space is too large and the evaluation of a single candidate is too expensive. In this case, only a subset of the solutions can be evaluated.

Optimization problems can be either function optimization problems or combinatorial problems. The first class of problems can be divided in continuous optimization and discrete optimization problems. In continuous function optimization, the independent variables are real numbers whereas for discrete function optimization, the independent variables can only be chosen from a predefined set of allowed and somehow ordered numbers, for example $\{10, 20, 30, 40\}$.

In combinatorial optimization problems, the optimum sequence or combination of a fixed set of input values has to be found. Here, the input values are symbols and might not be connected or ordered, for example $\{apple, orange, strawberry\}$. An example of a combinatorial optimization problem is the classical Travelling Salesman Problem (TSP), where a sales agent needs to visit a predefined set of cities and return to base. The problem here is to find an optimal route, i.e. the route that connects all cities whilst having the shortest travel distance, by choosing the order in which the cities are visited.

2.5 Optimization Loop

Mathematical or *calculus-based* methods use known functional relationships between variables and objectives to calculate the optimum of the given system. Therefore, an exact mathematical model of the process must exist. Edelbaum [8] introduced the differentiation of calculus-based methods in *direct methods* and *indirect methods*.

Direct methods solve the optimization problem by iterative calculation and derivation of the error function and moving in a direction to the maximum slope gradient. Indirect methods solve the optimization problem in one step – without testing – by solving a set of equations (usually non-linear). These equations result from setting the derivative of the error function equal to zero. Both classes of methods are local in scope, i.e. they tend to find only local optima. Therefore, they are not robust. They depend on the existence of derivatives. Real problem functions tend to be perturbed by noise and are not smooth, i.e. derivations may not exist for all points of functions. This class of problem cannot be solved by mathematical methods.

If the functional relations between input variables and objectives are not known, one can experiment on the real system (or a model of this system) in order to find the optimum. Access to the independent variables must exist for the whole multi-dimensional search space, i.e. the collection of all possible candidate solutions. Also a possibility of measuring the independent variable and the objective must be given. The optimization process is iterative, i.e. it has to be done in a closed *optimization loop* (Figure 3).

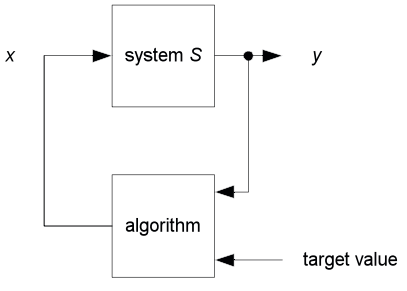

Fig. 3 Closed optimization loop consisting of a system and an optimization algorithm.

Experimental optimization methods can therefore be seen as a search for the optimum by traversing over the fitness landscape.

3 Genetic Algorithms

As Darwin's theory of natural selection articulates, nature is very effective at optimization, e.g. to enable life-forms to survive in a unfriendly and changing

environment, only by means of the simple method of trial and error. Genetic Algorithms (GAs) simulate this evolutionary mechanism by using *heredity* and *mutation*. They were first introduced in 1975 by Holland [11] who also provided a theoretical framework for Genetic Algorithms, the *Schemata Theorem* [13].

For genetic algorithms, the independent input parameters of a system *S* (Figure 4) are coded into a binary string, the *genotype* of an individual (Figure 5).

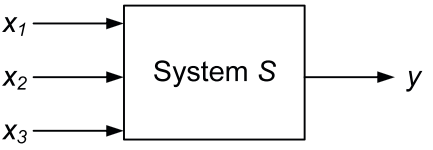

Fig. 4 System to be optimized.

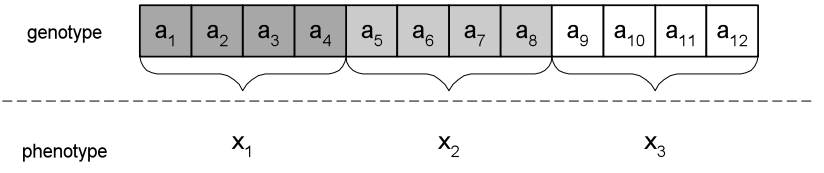

Fig. 5 Binary string representing one input pattern of the system.

The individual represented by genotype is called a *phenotype*. This phenotype has a certain quality or *fitness* to survive which can be determined by presenting the phenotype to the system *S* and measuring the system response.

The search is not only undertaken by one individual but by a population of n genotypes, the genepool. Therefore, the search space is tested at n points in parallel. All the individuals of the genepool at a time t_n are called a *generation*.

A new generation for time t_{n+1} is generated by selecting N individuals from the current population for breeding. They are copying into the genepool of the next generation and their genetic information is then recombined, using the *cross-over* operator (see 3.2), with a predefined cross-over probability p_c. The resulting offspring is then copied into the new genepool and mutation is applied to the offspring. Figure 7 shows the flowchart of a simple Genetic Algorithm.

The search will be carried out until at least one individual has a better fitness than the defined minimum fitness, or a maximum number of generations have been reached.

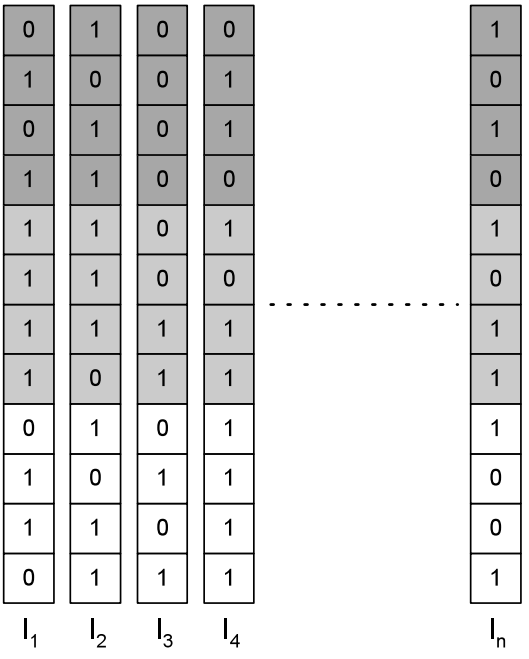

Fig. 6 Genepool consisting of individuals $I_1, \ldots, I_n$.

3.1 Selection

In general, there are three different approaches to choose individuals from the current generation for re-production, namely *Tournament Selection*, *Fitness Proportional Selection* and *Rank Based Selection*. In Tournament Selection, two or more individuals are randomly selected from the current generation of N genotypes to compete with each other. The individual with the highest fitness of this set is the winner and will be selected for generating offspring. The process is repeated N times in order to create the new population. Using Tournament Selection, the least fit individual can never be selected.

In fitness proportional selection, the chance of an individual to be selected is related to its fitness value. The most commonly used method of this type is Roulette Wheel Selection. Here, proportions of an imaginary roulette wheel are distributed in proportion to the relative fitness of an individual. Figure 8 shows and example for $N = 3$. In this example, the fitness of individual 3 is approximately four times higher than the fitness of individual 1, i.e. its chance to be selected is four times greater then the chance that individual one is selected. For a population of N individuals, the wheel is spun N times and the individual under the pointer is selected. In fitness proportional selection, all individuals have a chance of selection but high fitness individuals are more likely to be selected, because they occupy a larger portion of the wheel.

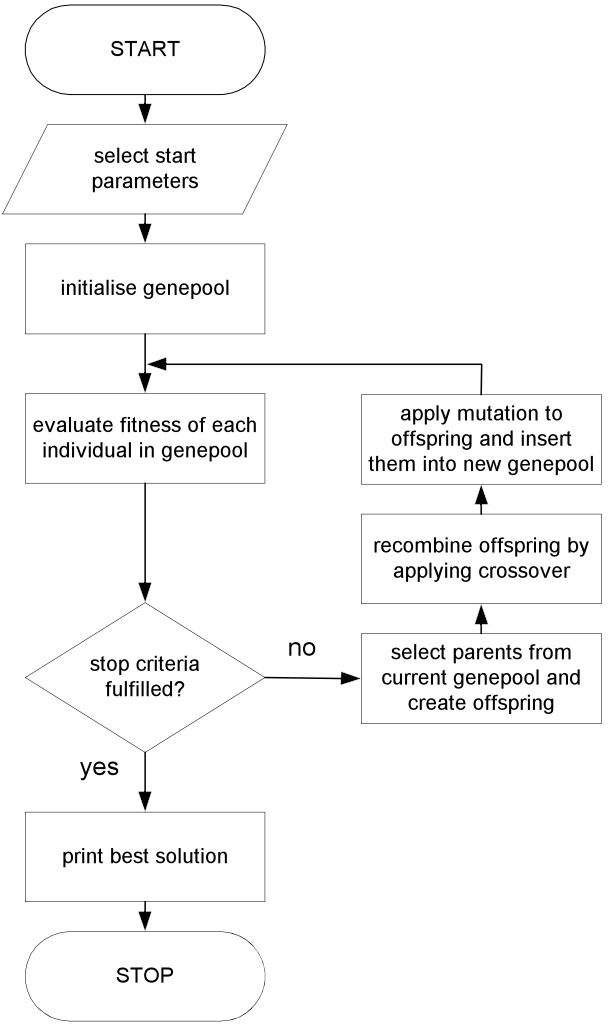

Fig. 7 Flowchart of basic GA algorithm.

However, there is the statistical chance that the actual selected distribution might not reflect the expected distribution based on the fitness values. If the selection is too strong, it can lead to premature convergence, i.e. the population would converge before it has found the region of the search space that contains the global optimum. In other words, the exploitation would start before the search space is fully explored. On the other hand, if the selection is too weak, it can lead to stalled evolution, which means the search is reduced to randomly walking through search space.

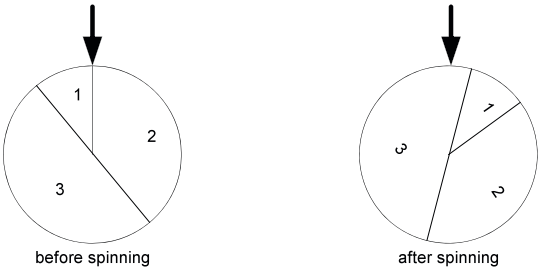

Fig. 8 Roulette Wheel selection.

These effects are overcome using Stochastic Universal Selection (SUS). Here, the same roulette wheel is used, but instead of using a single pointer, N equally-spaced pointers are used for a population of N individuals and the wheel is spun only once (Figure 9).

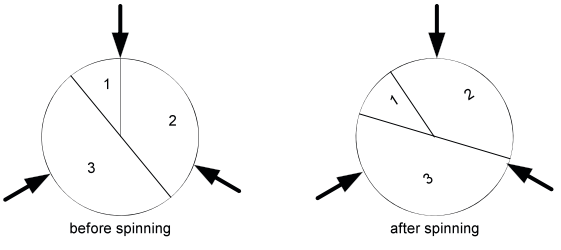

Fig. 9 SUS selection.

Instead of using the fitness of an individual for selection, a selective s value can be used, which is based on the rank position of an individual in the population (Equation 4).

$$s_i = Min + (Max - Min)\frac{rank_i - 1}{N - 1} \tag{4}$$

where

Min: minimum fitness within a generation
Max: maximum fitness within a generation
$rank_i$: rank of individual i within the population in a generation
N: number of individuals within population

So, instead of using the raw fitness to determine the proportion for an individual, the rank of the individual within the generation is used.

Sometimes the *m* fittest individuals in a generation are cloned into the next generation in order to make sure to preserve their genetic material. This is known as *elitism*.

3.2 Cross-Over

The most important operator in terms of robustness of the algorithm is the cross-over operator. Figure 10 shows the so-called *one-point cross-over* operator, which combines the information of two parents. They are aligning and then both cut at a randomly chosen cross-over point and the tails are swapped successively.

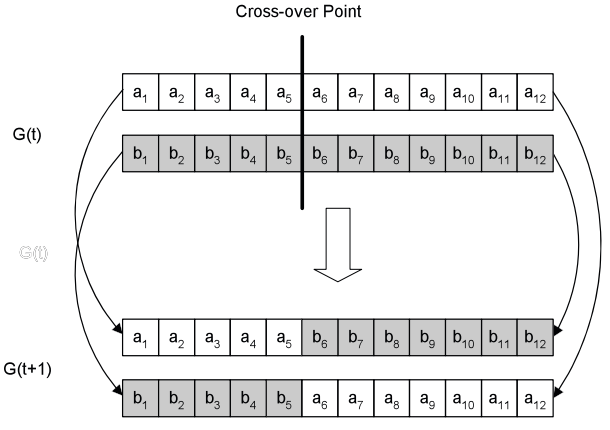

Fig. 10 Cross-over operator.

Instead of a single cross-over point, two or more random cross-over points can be used for recombining the genetic information of the parents.

Another form of cross-over is called *Uniform cross over* [22]. Here, every component of a parent individual X is randomly passed on either to offspring A or offspring B. If X passes on its component to A, the position in B is filled using the component from parent Y and vice versa.

3.3 Mutation

After the genetic information of the parents is recombined using cross-over, mutation is applied to every individual of the new generation. Here, every bit of the offspring is inverted (*mutated*) with probability p_m. The mutation operator is important for restoring lost information and therewith to result in a better effectiveness of the GA.

3.4 Discussion

The advantages of GAs are that they use payoff (objective function) information, not derivatives or other auxiliary knowledge, i.e. they are black box optimization methods. GAs tend to converge towards the global optimum rather than getting stuck in a local optimum and therefore they are very robust. On the other hand, it is not always straightforward to find the right GA parameters for a particular optimization problem, e.g. a suitable genepool size or mutation probability. Also, the efficiency of GAs relies heavily on the right coding of the input parameters, i.e. the chosen mapping function from phenotype to genotype, and they tend to fail if the inputs of the system are heavily correlated.

3.5 Schemata Theorem

Holland provided a theoretical foundation of GAs, i.e. a theoretical proof of convergence, which he called the *Schemata Theorem*. A schema is a template for binary strings, but built from a three letter alphabet containing the symbols *, 0 and 1. The * symbol is the 'dont care symbol' which either stands for 0 or 1. Figure 11 shows an example of a schema for chromosomes consisting of 12 bits, of which three are set to the dont care symbol and the remaining nine bits are set to fixed values.

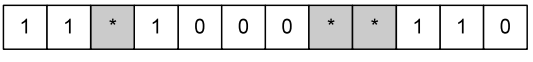

Fig. 11 Example of a schema in GA.

The distance between the first and the last fixed bit is called the *defined length* of the schema and the number of fixed bits is called the *order* of the schema. Figure 12 shows an example of a schema H and the different instances it represents.

A binary string s is an instance of a schema H if it fits into the template. Therefore, any binary string of length l does not just represent one candidate solution, it is also an instance of 2^l schemata at the same time. As a consequence, a GA with the genepool of size n does not only test n different solutions at the same time, but also a high number of different schemata. This is known as *implicit parallelism* in GA and provides an explanation for their effectiveness and efficiency.

According to Holland, the number of instances m of a schema H that are contained in the population at generation $t+1$ can be determined as follows:

$$m(H,t+1) = m(H,t) \cdot \frac{\bar{f}(H)}{\bar{f}} \tag{5}$$

where

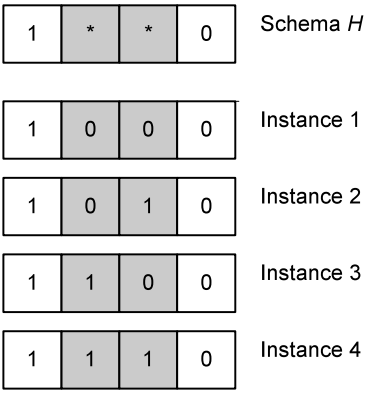

Fig. 12 Example of a schema H and the instances it represents.

H:	Schema or "Building Block" with at least one instance in the last generation,
$m(H,t)$:	number of instances of H at time t,
$m(H,t+1)$:	number of instances of H at time $t+1$,
$\bar{f}(H)$:	average fitness of the instances of schema H,
$\bar{f}$:	average fitness of the whole population.

This is a simplified version of the schemata theorem, because it does not take into account the effects of the cross-over and the mutation operator. However, it is sufficient to demonstrate the basic idea. A more detailed description can be found, for example, in the book of Goldberg [13].

Suppose that a particular schemata H remains above-average an amount $c \cdot \bar{f}$ with c being a constant factor, equation 4 can be rewritten as follows:

$$m(H,t+1) = m(H,t) \cdot \frac{\bar{f} + c \cdot \bar{f}}{\bar{f}} = (1+c) \cdot m(H,t) \tag{6}$$

Assuming c is stationary and starts at $t = 0$, equation 5 can be rewritten as follows:

$$m(H,t) = m(H,0) \cdot (1+c)' \tag{7}$$

It can be seen that this equation is similar to the formula of interest: the number of instances of a schema H with a fitness above-average grows exponentially to generation t. Hence, schemata with good fitness will survive and ones with a fitness below average will eventually die out. Therefore, the fitter building blocks, i.e. the better partial solution, will take over the genepool within finite time. However, the schemata theorem is controversial, because it assumes that the factor c is constant over time.

3.6 Coding Problem

Traditionally, GAs use binary stings. However, if an input variable is coded using standard binary coding, this can lead to the problem that a small change in the phenotype would require a large number of bits of the genotype to be inverted. An example of the coding problem is given in Figure 13.

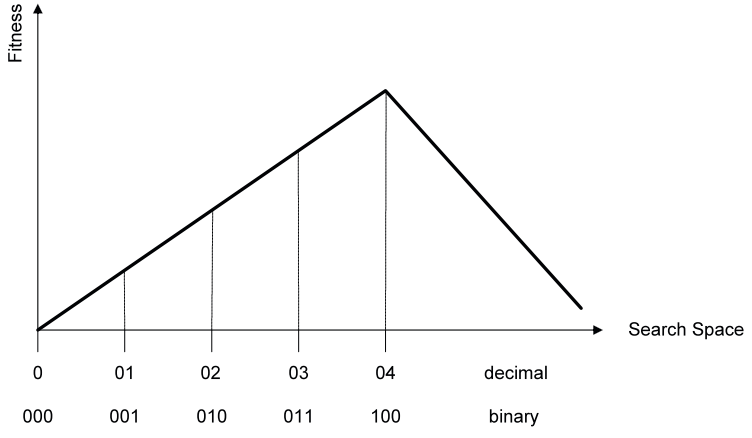

Fig. 13 Differences between decimal and standard binary coding.

As it can be seen from the figure, a step from 3_{10} to 4_{10} requires flipping 3 bits in binary representation whereas it only changes the least significant digit in decimal representation by one. One solution is to use Gray Code, which has the advantage that only one bit changes between any two positions, i.e. it has a constant Hamming Distance of one.

4 Ant Colony Optimization

Ant Colony Optimization (ACO) [4] refers to a class of discrete optimization algorithms, i.e. a meta-heuristic, which is modelled on the collective behaviour of ant colonies.

Real ants are very limited in their individual cognitive and visual capabilities, but an ant colony as a social entity is capable of solving complex problems and tasks in order to survive in an ever-changing hostile environment. For example, ants are capable of finding the shortest path to a food source [14]. If the food source is depleted, the ant colony adapts itself in a way that it will explore the environment and discover new food sources.

Ants communicate indirectly with other ants by depositing a substance called pheromone on the ground while they are walking around. This pheromone trail can then be used by the ant to find its way back to its nest after the ant has found a food

Decimal	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111

Fig. 14 Gray Code.

source and other ants can also sense it. Ants have the tendency to follow existing paths with high pheromone levels. If there is no existing pheromone trail, they walk around in a random fashion. If an ant has to make a decision, for example to choose a way around an obstacle in its way, it follows existing paths with a high probability. However, there is always a chance that the ant explores a new path or a path with a lower pheromone level. If an ant has chosen an existing path, the pheromone level of this path will be increased because the ants deposit new pheromone on top of the existing one. This makes it more likely that other ants will also follow this path, increasing the pheromone level again. This positive feedback process is known as autocatalysis [5]. Although the pheromone evaporates over time, the entire colony builds up a complex solution based on this indirect form of communication, called stigmergy [4].

Figure 15 demonstrates the basic principle of the ACO meta-heuristic, which is modelled after the behaviour described above. In this example, the system S that has to be optimized has three independent variables $x_1 \ldots x_3$ and the quality of the solution can be measured by the achieved fitness value y. Each input can have one of five different discrete alternative values s_{ij}, where i represents the input and j the chosen alternative for that input. Each alternative has an associated probability value, which is randomly initialised. The collection of probability distributions can be seen as a global probability matrix. Each artificial ant in the colony has to choose randomly a 'path' through state space, i.e. the input value for each independent variable. In the example in Figure 15, the ant chooses alternative s_{12} for input x_1, s_{24} for input x_2 and s_{33} for input x_3. The chosen path depends on the probabilities associated with the states, i.e. a state with a high probability is more likely to be selected for a trial solution than states with a low probability value. This probability values are refereed to as the pheromone level τ.

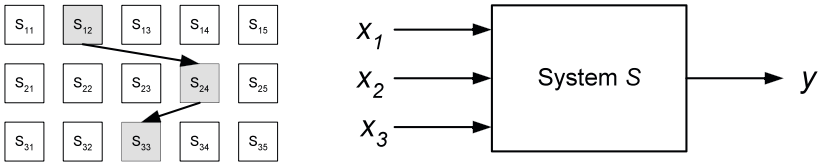

Fig. 15 Example of an artificial ant constructing a trial vector by traversing through state space.

A chosen path represents one candidate solution, which is evaluated and the probabilities of the states that the ant has visited on that trail is updated based on the achieved fitness. In the next generation, the updated probability matrix is used, which means that states that have proven fit in the past are more likely to be selected for the subsequent trail. However, it should be pointed out that a 'path' is not actually traversing through the state space; it simply refers to the collection of chosen alternatives for a particular candidate solution. The order in which the states are selected does not have any effect on the candidate solution itself, i.e. one could start with determining the input for x_1 first or, alternatively, with x_2 or x_3. The resulting candidate solutions would still be the same.

A major advantage of ACO is that adjacent states in the neighbourhood do not need to show similarities, i.e. the state space does not need to be ordered. This is different to most optimization heuristics, which rely on ordered collections of states, i.e. fitness landscapes.

Figure 16 shows a flowchart of the basic ACO meta-heuristic for a colony consisting of n artificial ants. During one iteration, called a time-step, every ant generates a trial solution, which is evaluated and based on the fitness of the solution the pheromone level of the states involved in the trail is updated in a local probability matrix for the ant. After one iteration, i.e. time-step, all the local probability matrices are combined and added to the global one, which is usually scaled down in order to simulate the evaporation process of real pheromone trails. This helps to avoid search stagnation and ensures that ants maintain their ability to explore new regions of the state space.

The main principle of ACO is that a colony of artificial ants builds up discrete probability distributions for each input parameter of a system to be optimized. Figure 17 shows an example of a probability distribution for an input i with ten alternative states.

It can be seen that state s_{i8} has the highest pheromone level, i.e. probability, and hence has a high chance to be selected for a trial. States s_{i7} and s_{i10}, on the other hand, have a pheromone level of zero and can never be selected. However, even states with a low pheromone level, e.g. s_{i3} in Figure 17, have a certain chance to be selected.

Initially, every possible choice for each of the input variables is set to a very low probability, which is the equivalent to the pheromone level in the real world. Each individual ant then chooses randomly one value for each input parameter, i.e. builds up a candidate solution, based on the probability distributions of the input values.

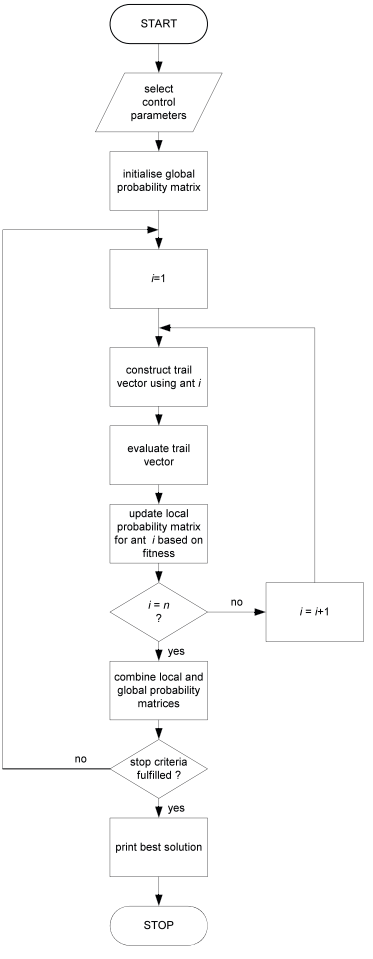

Fig. 16 Flowchart of ACO meta-heuristic.

Depending on the quality of the resulting candidate solution, the probability values of the chosen input values are updated. The whole process is repeated in iterations called time-steps until a suitable solution is found or the algorithm has converged, i.e. has reached a stable set of probability distributions. It has been proved, for example by Stützle and Dorigo [19] and Gutjahr [15], that ACO algorithms are capable of converging towards the global optimum within finite time.

The first computational optimization algorithm based on ant colonies was the Ant System (AS) algorithm [2]. It was successfully applied to the Travelling Salesman Problem and the Quadratic Assignment Problem. This was later followed by the Ant Colony System (ACS) [6], the Max-Min Ant System (MMAS) [20] and the Rank-Based Ant System (RBAS) [1].

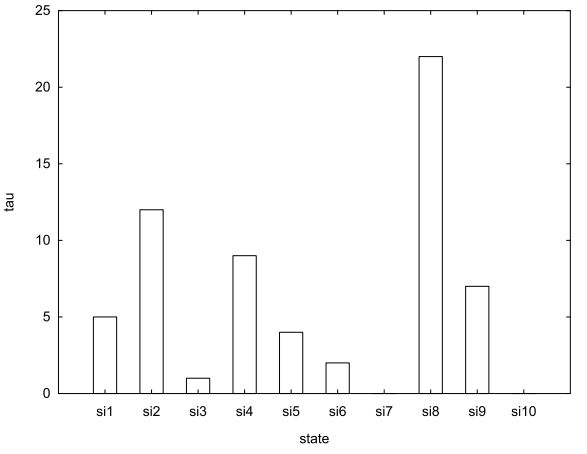

Fig. 17 Example of a probability distribution based on pheromone level.

For ACO the probability of a state s_{ij} to be chosen as input parameter i can be calculated using the following transition rule (Equation 8):

$$p(s_{ij}) = \begin{cases} \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{j=1}^{m} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}} & \text{if } s_{ij} \in N_i \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Where τ_{ij} is the pheromone level for state s_{ij}, η_{ij} is a heuristic value related to the fitness of the solution, α and β are control parameters that determine the relative importance of pheromone versus fitness, m is the number of alternatives for input parameter i, and N_i is the set of possible alternatives for input i. If the heuristic value η_{ij} is set to a constant value of one, the algorithm becomes the Simple Ant Colony Optimization algorithm (SACO) [7].

The evaporation after time-step t can be computed using Equation 9, where $\rho \in (0,1])$ is the evaporation rate.

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) \tag{9}$$

The pheromone updating rule is given in Equation 10, with $\Delta\tau_{ij}(t) = f(y_1, y_2, \ldots, y_n)$:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{10}$$

Unlike real ants, artificial ants can be equipped with additional capabilities, for example with look ahead capabilities [17] and backtracking [3] in order to improve efficiency. They can also be combined with local search methods [4, 18].

However, one problem related to ACO is that it is not a straightforward task to find optimum control parameter settings for an ACO application [12].

5 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a simple but effective algorithm that was original developed by Kennedy and Eberhart [16] for continuous function optimization. It is based on the social behaviour of a collection of animals that can be observed, for example, in fish schooling and bird flocking. PSO uses a population of agents where the population is referred to as swarm and the agents are called particles. Each particle represents an input vector for the system and is randomly initialised.

Each particle i has a position $x_{ij}(t)$ and a velocity $v_{ij}(t)$ for each dimension j of the search space. In every iteration of the algorithm, i is 'flying' through search space by adjusting the position vector $x_i(t)$ using the velocity vector $v_i(t)$ as follows:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{11}$$

It should be stressed that, in the physical world, a velocity and a position cannot be added. The velocity would need to be multiplied with a time interval in order to get a distance that could then be added to the original position. However, if one thinks of an iteration as a time step, the velocity vector could be multiplied with one time unit, which would not change the actual value but it would change the unit. The velocity vector itself is determined using the following equation:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_2(x_{i\_best} - x_{ij}(t)) + c_2 r_2(x_{global\_best} - x_{ij}(t)) \tag{12}$$

Here, r_1 and r_2 are random numbers, c_1 and c_2 are tuning constants, x_{i_best} is the best position that particle i found during the search so far and x_{global_best} is the best position the swarm found so far. The second term in Equation 12 is called the component cognitive component whereas the third one is called the social component. Figure 18 shows a flow-chart of the basic PSO algorithm.

One variation of the basic PSO algorithm is that, instead of using the global best position, the best position of the neighbourhood of particle i is used in the social component.

6 Overview

The single chapters of this book will demonstrate the practical application of the before-mentioned meta-heuristic optimization techniques. It can be seen how the pure academic perspective taken so far conveys into the complex aspects of an engineering application.

The chapter "Learning Automata in Control Planning Strategies" explores the use of planning in order to improve the performance of feedback-based control schemes considering only one probabilistic approach known as Learning Automata (LA). Authors propose a novel scheme that has been motivated by the human ability of choosing among different alternative plans while solving daily-life problems. Considering that Planning provides a very general and easy applicable methodology, the overall approach combines the use of Model predictive Control as regulation

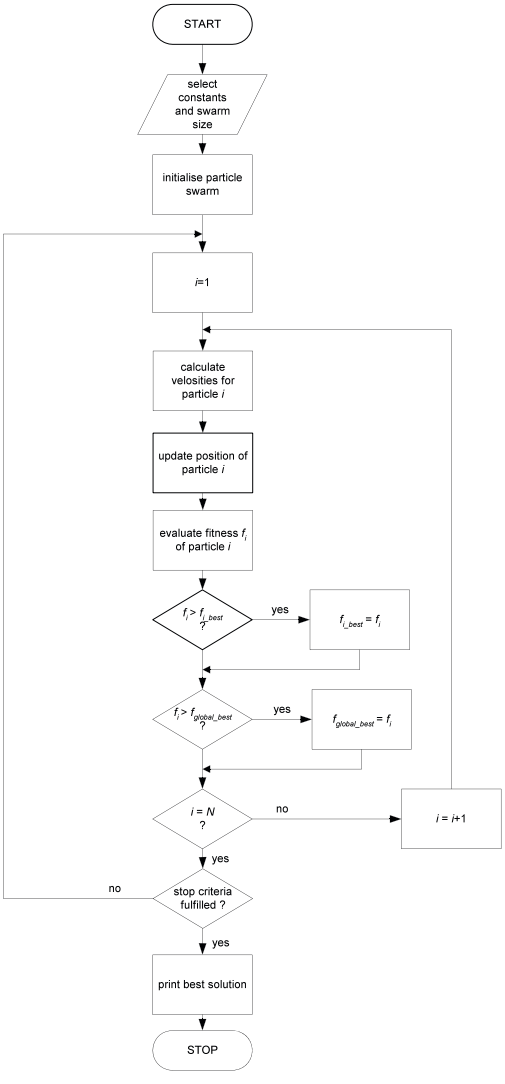

Fig. 18 Flowchart of PSO algorithm.

strategy and LA as optimization technique. The latter provides remarkable capabilities for global optimization on multimodal surfaces. By using LA, the search for the optimum is done over a probability space rather than exploring the parameter domain as it is commonly done by traditional algorithms. Some experiments and comparisons are conducted over a conventional control plant in order to demonstrate the proposed framework's performance.

In the chapter "Optimization Strategies for Restricted Candidate Lists in Field Service Scheduling," authors formally define a real-world class of combinatorial

optimization problems called Field Service Scheduling (FSS), and position it in relation to some optimization problems more often encountered in the literature. The presented research was motivated by the challenges encountered at routinely solving large instances of these problems in an industrial setting. The authors propose a generic framework that combines the expressiveness to address a wide variety of such problems with the ability to tune the optimization process to the type of instances being solved. Stategies for restricted candidate lists are introduced as a way to accelerate the pace of convergence when particular problem features are present. Two such strategies, constrained clustering and bundling, are explored in some detail, yielding some surprising results and material for further research. The presented experimental results were obtained on top of a GRASP meta-heuristic, but the presented approach of using strategies does not depend in any meaningful way on the particular meta-heuristic in use and can be generalized to other optimization methods.

The chapter "Framework for Integrating Optimization and Heuristic Models for Solving Planning and Scheduling Problem in a Resin Manufacturing Plant" describes a methodology that combines optimization and heuristic models to provide a solution to the planning and scheduling problem in a batch chemical plant. Real-life industrial data from a resin manufacturing plant was used to validate and test the robustness of the proposed methodology. The results of the proposed methodology are encouraging and provide substantial benefits to practitioners.

The chapter "Evolutionary Algorithms in the Optimal Sizing of Analog Circuits" highlights the application of two multi-objective evolutionary algorithms (NSGA-II and MOEA/D) in the optimization of analog integrated circuits. The algorithms use the circuit simulator SPICE to evaluate performances of unity-gain cells, current conveyors and CFOA. The results on the sizing process show that the genetic operator known as differential evolution increases the dominance of both evolutionary algorithms. Finally, some heuristics regarding the effectiveness of NSGA-II and MOEA/D in the sizing of analog integrated circuits are summarized.

Managing the core of a nuclear reactor so as to maximise the energy produced whilst meeting all of the safety requirements is a difficult and complex task, which is the topic of the chapter "Application of Estimation of Distribution Algorithms for Nuclear Fuel Management." A reactor core consists of many vertical channels into which a tube, containing the fuel, can be placed. Typically the fuel in each tube is different, so that one needs to place N different objects into each of N locations. This is a difficult problem in combinatorial optimization. The best available algorithm for this problem, a genetic algorithm with a specially designed crossover operator, was created nearly twenty years ago. In this research, an estimation of distribution algorithm supplemented with heuristic information to tackle the problem was used. The estimation of distribution algorithm produces a probability distribution function for each channel that indicates which fuel tubes are likely to be found in that channel in an optimal solution. Conversely it indicates which fuel tubes will not be found in the channel. The algorithm starts with a uniform probability distribution, i.e. all solutions are equally likely. The distribution is then sampled and the proposed solutions tested, this then allows the probability distribution to be updated.

The process can then be repeated until the algorithm converges onto an invariant probability distribution function. This algorithm has significantly outperformed the previous state-of-the-art method and represents a major improvement in the ability to tackle this problem.

The chapter "Optimal Control Systems with Reduced Parametric Sensitivity Based on Particle Swarm Optimization and Simulated Annealing" offers the design of optimal control systems with a reduced parametric sensitivity on the basis of Particle Swarm Optimization (PSO) and Simulated Annealing (SA) algorithms that belong to the popular category of nature-inspired optimization algorithms. PSO and SA algorithms are employed in solving optimization problems that optimize the control system responses and reduce the sensitivity with respect to parametric variations of the controlled process. The objective functions are expressed as integral quadratic performance indices that depend on the control error and on the squared output sensitivity functions. The PSO-based and SA-based minimization of the objective functions has as direct result the optimal tuning parameters of the controllers. The new optimization problems use the advantages of nature-inspired optimization algorithms to improve the control systems performance indices. This chapter contains recommendations for the practitioners that contribute to practical implementations with good computational efficency and fast convergence rate.

A comparative study of ant colony, differential evolution, particle swarm optimization and the neighbourhood algorithms for history matching of reservoir simulation models is provided in the chapter "Comparison of Evolutionary and Swarm Intelligence Methods for History Matching and Uncertainty Quantification in Petroleum Reservoir Models." In history matching, simulation models are calibrated to reproduce the historical observations from the oil and gas fields. History matching is an inverse problem with non-unique solution. Multiple history matched reservoir models are used to quantify uncertainty of future hydrocarbon production from a field. In our assisted history matching workflow, different evolutionary and swarm intelligence algorithms are use to explore the plausible parameter space and find good-fitting reservoir models. These algorithms are also integrated within a Bayesian framework to quantify uncertainty of the predictions. Two petroleum reservoir cases illustrate different aspects of this comparative study. The results present comparison of best history-matched models, convergence speeds for different algorithms, ability of the algorithms in navigating the search space and their effect on uncertainty of the predictions for ultimate oil recovery.

A contribution from the general field of logistics can be found in the chapter "Optimization of Multiple Traveling Salesmen Problem by a Novel Representation based Genetic Algorithm." The aim of logistics is to get the right materials to the right place at the right time, while optimizing a given performance measure and satisfying a given set of constraints. In most distribution systems goods are transported from various origins to various destinations. It is often economical to consolidate the shipments of various origin-destination pairs and transport such consolidated shipments in the same truck at the same time. Obviously the challenge is to find the optimal i.e. the best consolidation according to some objective functions.

In case of this chapter, the problem is an asymmetric multiple Traveling Salesman Problem with Time Windows, where additional special constraints exist, like an upper bound for the number of salesmen, the maximum travelling distance, or a time window at each location. This is a numerical optimization problem, obviously an NP-hard task.

The main motivation of the research presented in this chapter was that there was no available algorithm that is "intelligent" enough to handle constraints on tour lengths, asymmetric distances, and the number of salesmen is not predefined, and can vary during the evolution of the individual solutions. Furthermore the representation is so transparent that supports not only the implementation, but the initialization and heuristic fine-tuning of the individual routes.

In this chapter, authors propose a general, novel genetic representation for the so-called mTSP problems. A new genetic algorithm based on the novel representation is presented, which can handle the constraints for the routes and the time windows for the locations too, as well as its representation is more similar for the characteristic of the problem than the ones until now. Thus it can be more easily understandable and realizable. The algorithm was implemented in MATLAB and integrated with Google Maps to provide a complete framework for distance calculation, definition of the initial routes and visualization of the resulted solutions. The novel approach and the novel representation proved to be more effective in terms of flexibility, complexity and transparency, and also in efficiency than the previous methods.

In the chapter "Out-of-the-box and Custom Implementation of Metaheuristics. A Case Study: The Vehicle Routing Problem with Stochastic Demand" authors propose an experimental analysis that studies the impact of development effort on the relative performance of metaheuristics. Five algorithms for the vehicle routing problem with stochastic demand that have been proposed in the literature are considered: Tabu Search, Simulated Annealing, Genetic Algorithms, Iterated Local Search and Ant Colony Optimization. As measure of the development effort the time devoted to tune the parameters of the algorithms is considered. In this way, such effort can be easily measured. If such a minor implementation issue allows to point out a significant difference in the relative behavior of metaheuristics, then this difference can be expected to grow when one considers other issues. The same algorithms in their out-of-the-box version, i.e., with no parameter tuning, and in their custom one, i.e., with fine-tuned parameters are compared. The results support the main claim of the chapter: one should clearly state in which context one develops algorithms, since the results obtained in an out-of-the-box context are not necessarily extendable to a custom one, and vice versa. Moreover, in experimental analysis we often consider also the values of parameters indicated in the paper in which the algorithm was proposed. This analysis allows one to observe that the results that are reported in the literature cannot be a priori related to one of the two contexts.

Optimal analogue circuit sizing is investigated in the chapter "Analogue Circuit Optimization Through a Hybrid Approach." It is shown that hybridization of a global optimization approach with a local one leads to better results in optimization of such circuits than using classical approaches. The case of merging Genetic

Algorithms with the Simulated Annealing technique is considered. The hybrid algorithm is detailed and is evaluated using test functions. It is shown through three application examples, i.e. optimization of performances of a current conveyor, an operational transconductance amplifier and a low noise amplifier, that such hybrid algorithms yield optimal solutions in a much shorter time, when compared to conventional meta-heuristics.

The scope of the chapter "Evolutionary Inventory Control for Multi-Echelon Systems" is confined to the use of Genetic Algorithms (GA)s for handling operational issues of inventory control and management in multi-echelon inventory networks. It provides an extensive review of literature on use of GAs to solve multi-echelon inventory control problems and evaluates the state of GA applications in these areas. The chapter also presents a novel GA structure for a stochastic lot sizing problem in a centralized distribution system. Numerical experiments conducted on several test cases with different operational parameters show that proposed GA structure can be used as an effective algorithm for solving the multi-echelon inventory distribution problem under stochastic demand.

To improve the ride comfort is one of the most important design objectives in automotive engineering by reducing the vibration transmission and keeping proper tyre contact. This is the main topic of the chapter "Fuzzy Skyhook Surface Control using Micro-Genetic Algorithm for Vehicle Suspension Ride Comfort." The semi-active suspension systems are developed to achieve a better ride comfort performance than the passive suspension system. A polynomial function supervised fuzzy sliding mode control collaborated with a skyhook surface method is introduced for the ride comfort of a two degree of freedom vehicle semi-active suspension. The multi-objective micro-genetic algorithm has been utilised to this proposed controller's parameter alignment in a training process with three ride comfort objectives. The numerical results have shown that this hybrid control method is able to provide a real-time enhanced level of ride comfort performance for the semi-active suspension system.

References

1. Bullheimer, B., Hartl, R.F., Strauss, C.: A new rank-based version of the Ant System: A computational study. Central European Journal for Operations Research and Economics 7(1), 25–38 (1999)
2. Dorigo, M.: Optimization, Learning and Natural Algorithms. PhD Thesis, Politecnico di Milano, Italy (1992)
3. Di Caro, G., Dorigo, M.: AntNet: Distributed Stigmergetic Control for Communications Networks. Journal of Artificial Intelligence Research 9, 317–365 (1998)
4. Dorigo, M., Di Caro, G.D., Gambardella, L.M.: Ant Algorithms for Discrete Optimization. Artificial Life 5, 137–172 (1999)
5. Dorigo, M., Maniezzo, V., Colorni, A.: Positive Feedback as a Search Strategy, Technical Report No 91-016, Politecnico di Milano (1991)
6. Dorigo, M., Gambardella, L.: Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. IEEE Transactions on Evolutionary Computation 1(1), 53–66 (1997)

7. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
8. Edelbaum, T.N.: Theory of Maxima and Minima. In: Leitmann (ed.) Optimization Techniques with Applications to Aerospace Systems, p. 132. Academic Press, New York (1962)
9. Fenton, N., Hill, G.: Systems construction and analysis: a mathematical and logical framework. McGraw-Hill, New York (1993)
10. Fonseca, C.M., Fleming, P.J.: An Overview of Evolutional Algorithms in Multiobjective Optimization. Evolutionary Computation (31), 1–16 (1995)
11. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
12. Gaertner, D., Clark, K.: On Optimal Parameters for Ant Colony Optimization algorithms. In: Proceedings of the International Conference on Artificial Intelligence 2005, Las Vegas, USA, pp. 83–89 (2005)
13. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
14. Goss, S., Aron, S., Deneubourg, J.L., Pasteels, J.M.: Self-organized shortcuts in the Argentine ant. Naturwissenschaften 76, 579–581 (1989)
15. Gutjahr, W.: A graph-based ant system and its convergence. Future Generation Computer Systems 16(8), 873–888 (2000)
16. Kennedy, J., Eberhart, E.: Particle Swarm Optimization. In: IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948. IEEE Press, Los Alamitos (1995)
17. Michel, R., Middendorf, M.: An Island Model Based Ant System with Lookahead for the Shortest Supersequence Problem. In: Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature, Amsterdam, The Netherlands, pp. 692–701 (1998)
18. Shmygelska, A., Hoos, H.H.: An Ant Colony Optimisation Algorithm for the 2D and 3D Hydrophobic Polar Protein Folding Problem. BMC Bioinformatics 6, 6–30 (2005)
19. Stützle, T., Dorigo, M.: A short convergence proof for a class of ant colony optimisation algorithms. IEEE Transactions on Evolutionary Computation 6(4), 358–365 (2002)
20. Stützle, T., Hoos, H.H.: MAX-MIN Ant System. Future Generation Computer Systems 16(8), 889–914 (2000)
21. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Son, Inc., New York (1995)
22. Syswerda, G.: Uniform Crossover in Genetic Algorithms. In: Proceedings of International Conference on Genetic Algorithm 1989 (ICGA 1989), pp. 2–9 (1989)
23. Wright, S.: Evolution in Mendelian populations. Genetics 16, 97–159 (1931)