1

Swarm Intelligence: Foundations, Perspectives and Applications

Ajith Abraham[1], He Guo[2], and Hongbo Liu[2]

[1] IITA Professorship Program, School of Computer Science and Engineering, Chung-Ang University, Seoul, 156-756, Korea. ajith.abraham@ieee.org, http://www.softcomputing.net

[2] Department of Computer Science, Dalian University of Technology, Dalian, 116023, China. {guohe,lhb}@dlut.edu.cn

This chapter introduces some of the theoretical foundations of swarm intelligence. We focus on the design and implementation of the Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) algorithms for various types of function optimization problems, real world applications and data mining. Results are analyzed, discussed and their potentials are illustrated.

1.1 Introduction

Swarm Intelligence (SI) is an innovative distributed intelligent paradigm for solving optimization problems that originally took its inspiration from the biological examples by swarming, flocking and herding phenomena in vertebrates.

Particle Swarm Optimization (PSO) incorporates swarming behaviors observed in flocks of birds, schools of fish, or swarms of bees, and even human social behavior, from which the idea is emerged [14, 7, 22]. PSO is a population-based optimization tool, which could be implemented and applied easily to solve various function optimization problems, or the problems that can be transformed to function optimization problems. As an algorithm, the main strength of PSO is its fast convergence, which compares favorably with many global optimization algorithms like Genetic Algorithms (GA) [13], Simulated Annealing (SA) [20, 27] and other global optimization algorithms. For applying PSO successfully, one of the key issues is finding how to map the problem solution into the PSO particle, which directly affects its feasibility and performance.

Ant Colony Optimization (ACO) deals with artificial systems that is inspired from the foraging behavior of real ants, which are used to solve discrete

A. Abraham et al.: *Swarm Intelligence: Foundations, Perspectives and Applications*, Studies in Computational Intelligence (SCI) **26**, 3–25 (2006)
www.springerlink.com

optimization problems [9]. The main idea is the indirect communication between the ants by means of chemical pheromone trials, which enables them to find short paths between their nest and food.

This Chapter is organized as follows. Section 1.2 presents the canonical PSO algorithm and its performance is compared with some global optimization algorithms. Further some extended versions of PSO is presented in Section 1.3 followed by some illustrations/applications in Section 1.4. Section 1.5 presents the ACO algorithm followed by some illustrations/applications of ACO in Section 1.6 and Section 1.7. Some conclusions are also provided towards the end, in Section 1.8.

1.2 Canonical Particle Swarm Optimization

1.2.1 Canonical Model

The canonical PSO model consists of a swarm of particles, which are initialized with a population of random candidate solutions. They move iteratively through the d-dimension problem space to search the new solutions, where the fitness, f, can be calculated as the certain qualities measure. Each particle has a position represented by a position-vector $\mathbf{x}_i$ (i is the index of the particle), and a velocity represented by a velocity-vector $\mathbf{v}_i$. Each particle remembers its own best position so far in a vector $\mathbf{x}_i^{\#}$, and its j-th dimensional value is $x_{ij}^{\#}$. The best position-vector among the swarm so far is then stored in a vector $\mathbf{x}^*$, and its j-th dimensional value is x_j^*. During the iteration time t, the update of the velocity from the previous velocity to the new velocity is determined by Eq.(1.1). The new position is then determined by the sum of the previous position and the new velocity by Eq.(1.2).

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(x_{ij}^{\#}(t) - x_{ij}(t)) + c_2 r_2(x_j^*(t) - x_{ij}(t)). \quad (1.1)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1). \quad (1.2)$$

where w is called as the inertia factor, r_1 and r_2 are the random numbers, which are used to maintain the diversity of the population, and are uniformly distributed in the interval [0,1] for the j-th dimension of the i-th particle. c_1 is a positive constant, called as coefficient of the self-recognition component, c_2 is a positive constant, called as coefficient of the social component. From Eq.(1.1), a particle decides where to move next, considering its own experience, which is the memory of its best past position, and the experience of its most successful particle in the swarm. In the particle swarm model, the particle searches the solutions in the problem space with a range $[-s, s]$ (If the range is not symmetrical, it can be translated to the corresponding symmetrical range.) In order to guide the particles effectively in the search space, the maximum moving distance during one iteration must be clamped in between the maximum velocity $[-v_{max}, v_{max}]$ given in Eq.(1.3):

$$v_{ij} = sign(v_{ij})min(|v_{ij}|, v_{max}). \tag{1.3}$$

The value of v_{max} is $p \times s$, with $0.1 \leq p \leq 1.0$ and is usually chosen to be s, i.e. $p = 1$. The pseudo-code for particle swarm optimization algorithm is illustrated in Algorithm 1.

Algorithm 1 Particle Swarm Optimization Algorithm

01. Initialize the size of the particle swarm n, and other parameters.
02. Initialize the positions and the velocities for all the particles randomly.
03. While (the end criterion is not met) do
04. $t = t + 1$;
05. Calculate the fitness value of each particle;
06. $\mathbf{x}^* = argmin_{i=1}^{n}(f(\mathbf{x}^*(t-1)), f(\mathbf{x}_1(t)), f(\mathbf{x}_2(t)), \cdots, f(\mathbf{x}_i(t)), \cdots, f(\mathbf{x}_n(t)))$;
07. For i= 1 to n
08. $\mathbf{x}_i^{\#}(t) = argmin_{i=1}^{n}(f(\mathbf{x}_i^{\#}(t-1)), f(\mathbf{x}_i(t))$;
09. For j = 1 to $Dimension$
10. Update the j-th dimension value of $\mathbf{x}_i$ and $\mathbf{v}_i$
10. according to Eqs.(1.1), (1.2), (1.3);
12. Next j
13. Next i
14. End While.

The end criteria are usually one of the following:

- Maximum number of iterations: the optimization process is terminated after a fixed number of iterations, for example, 1000 iterations.
- Number of iterations without improvement: the optimization process is terminated after some fixed number of iterations without any improvement.
- Minimum objective function error: the error between the obtained objective function value and the best fitness value is less than a pre-fixed anticipated threshold.

1.2.2 The Parameters of PSO

The role of inertia weight w, in Eq.(1.1), is considered critical for the convergence behavior of PSO. The inertia weight is employed to control the impact of the previous history of velocities on the current one. Accordingly, the parameter w regulates the trade-off between the global (wide-ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine-tuning the current search area. A suitable value for the inertia weight w usually provides balance between global and local exploration abilities and consequently results in a reduction of the number

of iterations required to locate the optimum solution. Initially, the inertia weight is set as a constant. However, some experiment results indicates that it is better to initially set the inertia to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined solutions [11]. Thus, an initial value around 1.2 and gradually reducing towards 0 can be considered as a good choice for w. A better method is to use some adaptive approaches (example: fuzzy controller), in which the parameters can be adaptively fine tuned according to the problems under consideration [24, 16].

The parameters c_1 and c_2, in Eq.(1.1), are not critical for the convergence of PSO. However, proper fine-tuning may result in faster convergence and alleviation of local minima. As default values, usually, $c_1 = c_2 = 2$ are used, but some experiment results indicate that $c_1 = c_2 = 1.49$ might provide even better results. Recent work reports that it might be even better to choose a larger cognitive parameter, c_1, than a social parameter, c_2, but with $c_1 + c_2 \leq 4$ [7].

The particle swarm algorithm can be described generally as a population of vectors whose trajectories oscillate around a region which is defined by each individual's previous best success and the success of some other particle. Various methods have been used to identify some other particle to influence the individual. Eberhart and Kennedy called the two basic methods as "gbest model" and "lbest model" [14]. In the lbest model, particles have information only of their own and their nearest array neighbors' best (lbest), rather than that of the entire group. Namely, in Eq.(1.4), gbest is replaced by lbest in the model. So a new neighborhood relation is defined for the swarm:

$$v_{id}(t+1) = w * v_{id}(t) + c_1 * r_1 * (p_{id}(t) - x_{id}(t)) + c_2 * r_2 * (p_{ld}(t) - x_{id}(t)). \quad (1.4)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1). \quad (1.5)$$

In the gbest model, the trajectory for each particle's search is influenced by the best point found by any member of the entire population. The best particle acts as an attractor, pulling all the particles towards it. Eventually all particles will converge to this position. The lbest model allows each individual to be influenced by some smaller number of adjacent members of the population array. The particles selected to be in one subset of the swarm have no direct relationship to the other particles in the other neighborhood. Typically lbest neighborhoods comprise exactly two neighbors. When the number of neighbors increases to all but itself in the lbest model, the case is equivalent to the gbest model. Some experiment results testified that gbest model converges quickly on problem solutions but has a weakness for becoming trapped in local optima, while lbest model converges slowly on problem solutions but is able to "flow around" local optima, as the individuals explore different regions. The gbest model is recommended strongly for unimodal objective functions, while a variable neighborhood model is recommended for multimodal objective functions.

Kennedy and Mendes [15] studied the various population topologies on the PSO performance. Different concepts for neighborhoods could be envisaged. It can be observed as a spatial neighborhood when it is determined by the Euclidean distance between the positions of two particles, or as a sociometric neighborhood (e.g. the index position in the storing array). The different concepts for neighborhood leads to different neighborhood topologies. Different neighborhood topologies primarily affect the communication abilities and thus the group's performance. Different topologies are illustrated in Fig. 1.1. In the case of a global neighborhood, the structure is a fully connected network where every particle has access to the others' best position (Refer Fig. 1.1(a)). But in local neighborhoods there are more possible variants. In the von Neumann topology (Fig. 1.1(b)), neighbors above, below, and each side on a two dimensional lattice are connected. Fig. 1.1(e) illustrates the von Neumann topology with one section flattened out. In a pyramid topology, three dimensional wire frame triangles are formulated as illustrated in Fig. 1.1(c). As shown in Fig. 1.1(d), one common structure for a local neighborhood is the circle topology where individuals are far away from others (in terms of graph structure, not necessarily distance) and are independent of each other but neighbors are closely connected. Another structure is called wheel (star) topology and has a more hierarchical structure, because all members of the neighborhood are connected to a 'leader' individual as shown in Fig. 1.1(f). So all information has to be communicated though this 'leader', which then compares the performances of all others.

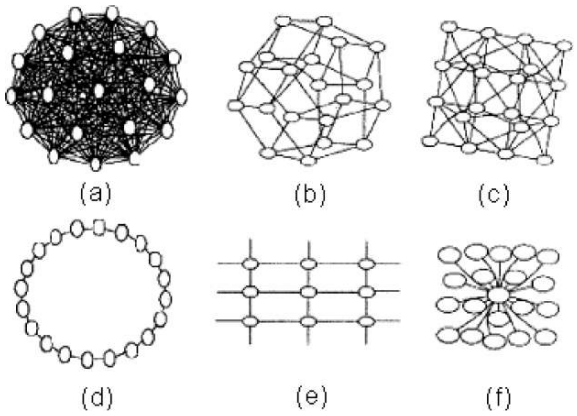

Fig. 1.1. Some neighborhood topologies adapted from [15]

1.2.3 Performance Comparison with Some Global Optimization Algorithms

We compare the performance of PSO with Genetic Algorithm (GA) [6, 13] and Simulated Annealing (SA)[20, 27]. GA and SA are powerful stochastic search and optimization methods, which are also inspired from biological and thermodynamic processes.

Genetic algorithms mimic an evolutionary natural selection process. Generations of solutions are evaluated according to a fitness value and only those candidates with high fitness values are used to create further solutions via crossover and mutation procedures.

Simulated annealing is based on the manner in which liquids freeze or metals re-crystalize in the process of annealing. In an annealing process, a melt, initially at high temperature and disordered, is slowly cooled so that the system at any time is approximately in thermodynamic equilibrium. In terms of computational simulation, a global minimum would correspond to such a "frozen" (steady) ground state at the temperature T=0.

The specific parameter settings for PSO, GA and SA used in the experiments are described in Table 1.1.

Table 1.1. Parameter settings for the algorithms.

Algorithm	Parameter name	Parameter value
GA	Size of the population	20
	Probability of crossover	0.8
	Probability of mutation	0.02
	Scale for mutations	0.1
	Tournament probability	0.7
SA	Number operations before temperature adjustment	20
	Number of cycles	10
	Temperature reduction factor	0.85
	Vector for control step of length adjustment	2
	Initial temperature	50
PSO	Swarm size	20
	Self-recognition coefficient c_1	1.49
	Social coefficient c_2	1.49
	Inertia weight w	$0.9 \rightarrow 0.1$

Benchmark functions:

- Griewank function:
  $f_1 = \frac{1}{4000}\sum_{i=1}^{n}(x_i)^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1$
  $\mathbf{x} \in [-300, 300]^n$, $min(f_1(\mathbf{x}^*)) = f_1(\mathbf{0}) = 0$.

- Schwefel 2.26 function:
  $f_2 = 418.9829n - \sum_{i=1}^{n}(x_i sin(\sqrt{|x_i|}))$
  $\mathbf{x} \in [-500, 500]^n$, $min(f_2(\mathbf{x}^*)) = f_2(\mathbf{0}) = 0$.
- Quadric function:
  $f_3 = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$
  $\mathbf{x} \in [-100, 100]^n$, $min(f_3(\mathbf{x}^*)) = f_3(\mathbf{0}) = 0$.

Three continuous benchmark functions, i.e. Griewank function, Schwefel 2.26 function and Quadric function, are used to test PSO, GA and SA. Quadric function has a single minimum, while the other two functions are highly multi-modal with multiple local minima. For all the test functions, the goal is to find the global minima. Each algorithm (for each function) was repeated 10 times with different random seeds. Each trial had a fixed number of 18,000 iterations. The objective functions were evaluated 360,000 times in each trial. The swarm size in PSO was 20, population size in GA was 20, the number operations before temperature adjustment in SA was set to 20. Figures 1.2, 1.3 and 1.4 illustrate the mean best function values for the three functions. It is observed that for GA and SA, the solutions get trapped in a local minimum even before 2000 iterations, for high dimensional, multi-modal functions, especially for Schwefel 2.26 function, while PSO performance is much better. For the Quadric function, SA performed well and PSO performance was comparatively poor, as depicted in Fig. 1.4.

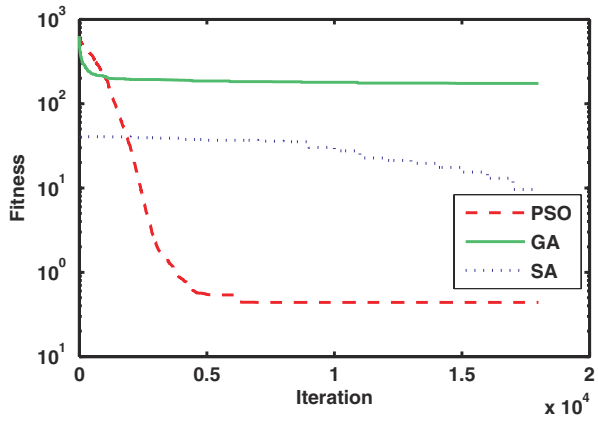

Fig. 1.2. Griewank function performance

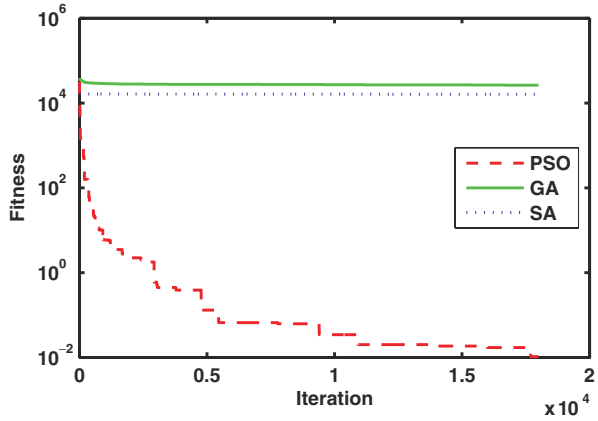

Fig. 1.3. Schwefel function performance

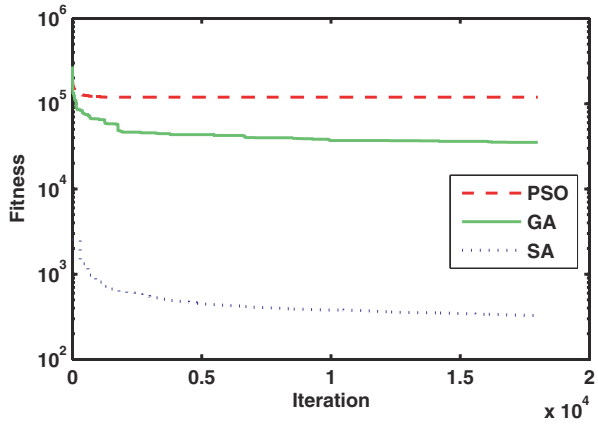

Fig. 1.4. Quadric function performance

1.3 Extended Models of PSO for Discrete Problems

1.3.1 Fuzzy PSO

In the fuzzy PSO model, the representations of the position and velocity of the particles in PSO are extended from real vectors to fuzzy matrices [21]. This is illustrated using the well known job scheduling problem. For a job scheduling problem: the jobs $J = \{J_1, J_2, \cdots, J_n\}$ are to be scheduled on the machines $M = \{M_1, M_2, \cdots, M_m\}$, and the fuzzy scheduling relation from M to J can be expressed as follows:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

where x_{ij} represents the degree of membership of the i-th element M_i in domain M and the j-th element J_j in domain J to relation X. The fuzzy relation X between M and J has the following meaning: for each element in the matrix X, the element

$$x_{ij} = \mu_R(M_i, J_j), i \in \{1, 2, \cdots, m\}, j \in \{1, 2, \cdots, n\}. \tag{1.6}$$

μ_R is the membership function, the value of x_{ij} means the degree of membership that M_j would process J_i in the feasible schedule solution. The elements of the matrix X should satisfy the following conditions:

$$x_{ij} \in [0, 1], i \in \{1, 2, \cdots, m\}, j \in \{1, 2, \cdots, n\}. \tag{1.7}$$

$$\sum_{i=1}^{m} x_{ij} = 1, i \in \{1, 2, \cdots, m\}, j \in \{1, 2, \cdots, n\}. \tag{1.8}$$

Similarly the velocity of the particle is defined as:

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{bmatrix}$$

The operators of Eqs.(1.1) and (1.2) should be re-defined because the position and velocity have been transformed to the form of matrices. The symbol "$\otimes$" is used to denote the modified multiplication. Let α be a real number, $\alpha \otimes V$ or $\alpha \otimes X$ means all the elements in the matrix V or X are multiplied by α. The symbols "$\oplus$" and "$\ominus$" denote the addition and subtraction between matrices respectively. Suppose A and B are two matrices which denote position or velocity, then $A \oplus B$ and $A \ominus B$ are regular addition and subtraction operation between matrices.

Then we obtain Eqs.(1.9) and (1.10) for updating the positions and velocities of the particles in the fuzzy discrete PSO:

$$V(t+1) = w \otimes V(t) \oplus (c_1 * r_1) \otimes (X^{\#}(t) \ominus X(t)) \oplus (c_2 * r_2) \otimes (X^*(t) \ominus X(t)). \tag{1.9}$$

$$X(t+1) = X(t) \oplus V(t+1)). \tag{1.10}$$

The position matrix may violate the constraints of Eqs.(1.7) and (1.8) after some iterations, so it is necessary to normalize the position matrix. First we make all the negative elements in the matrix become zero. If all elements

in a column of the matrix are zero, they need be re-evaluated using a series of random numbers with the interval [0,1]. And then the matrix undergoes the following transformation without violating the constraints:

$$Xnormal = \begin{bmatrix} x_{11}/\sum_{i=1}^{m} x_{i1} & x_{12}/\sum_{i=1}^{m} x_{i2} & \cdots & x_{1n}/\sum_{i=1}^{m} x_{in} \\ x_{21}/\sum_{i=1}^{m} x_{i1} & x_{22}/\sum_{i=1}^{m} x_{i2} & \cdots & x_{2n}/\sum_{i=1}^{m} x_{in} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1}/\sum_{i=1}^{m} x_{i1} & x_{m2}/\sum_{i=1}^{m} x_{i2} & \cdots & x_{mn}/\sum_{i=1}^{m} x_{in} \end{bmatrix}$$

Since the position matrix indicates the potential scheduling solution, we should "decode" the fuzzy matrix and get the feasible solution. A flag array could be used to record whether we have selected the columns of the matrix and a array to record the solution. First all the columns are not selected, then for each columns of the matrix, we choose the element which has the max value, then mark the column of the max element "selected", and the column number are recorded to the solution array. After all the columns have been processed, we get the feasible solution from the solution array and measure the fitness of the particles.

1.3.2 Binary PSO

The canonical PSO is basically developed for continuous optimization problems. However, lots of practical engineering problems are formulated as combinatorial optimization problems. The binary PSO model was presented by Kennedy and Eberhart, and is based on a very simple modification of the real-valued PSO. Faced with a problem-domain where we cannot fit into some sub-space of the real-valued n-dimensional space, which is required by the PSO, odds are that we can use a binary PSO instead. All we must provide, is a mapping from this given problem-domain to the set of bit strings. As with the canonical PSO, a fitness function f must be defined. In the binary PSO, we can define a particle's position and velocity in terms of changes of probabilities that will be in one state or the other, i.e. yes or no, true or false, or making some other decision. When the particle moves in a state space restricted to zero and one on each dimension, the change of probability with time steps is defined as follows:

$$P(x_{ij}(t+1) = 1) = f(x_{ij}(t), v_{ij}(t), x_{ij}^{\#}(t), x_j^{*}(t)). \tag{1.11}$$

where the probability function is usually

$$sign(v_{ij}(t+1) = 1) = \frac{1}{1 + e^{-v_{ij}(t)}}. \tag{1.12}$$

At each time step, each particle updates its velocity and moves to a new position according to Eqs.(1.13) and (1.14):

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(x_{ij}^{\#}(t) - x_{ij}(t)) + c_2 r_2(x_j^{*}(t) - x_{ij}(t)). \tag{1.13}$$

$$x_i(t+1) = \begin{array}{l} 1 \text{ if } \rho \leq s(v_i(t)), \\ 0 \text{ otherwise.} \end{array} \tag{1.14}$$

where c_1, c_2 are learning factors; w is inertia factor; r_1, r_2, ρ are random functions in the closed interval [0, 1].

1.4 Applications of Particle Swarm Optimization

1.4.1 Job Scheduling on Computational Grids

Grid computing is a computing framework to meet the growing computational demands. Essential grid services contain more intelligent functions for resource management, security, grid service marketing, collaboration and so on. The load sharing of computational jobs is the major task of computational grids [2].

To formulate our objective, define $C_{i,j}$ ($i \in \{1, 2, \cdots, m\}$, $j \in \{1, 2, \cdots, n\}$) as the completion time that the grid node G_i finishes the job J_j, $\sum C_i$ represents the time that the grid node G_i finishes all the scheduled jobs. Define $C_{max} = max\{\sum C_i\}$ as the makespan, and $\sum_{i=1}^{m}(\sum C_i)$ as the flowtime. An optimal schedule will be the one that optimizes the flowtime and makespan. The conceptually obvious rule to minimize $\sum_{i=1}^{m}(\sum C_i)$ is to schedule Shortest Job on the Fastest Node (SJFN). The simplest rule to minimize C_{max} is to schedule the Longest Job on the Fastest Node (LJFN). Minimizing $\sum_{i=1}^{m}(\sum C_i)$ asks the average job finishes quickly, at the expense of the largest job taking a long time, whereas minimizing C_{max}, asks that no job takes too long, at the expense of most jobs taking a long time. Minimization of C_{max} will result in the maximization of $\sum_{i=1}^{m}(\sum C_i)$.

To illustrate the performance of the algorithms, we considered a finite number of grid nodes and assumed that the processing speeds of the grid nodes (cput) and the job lengths (processing requirements in cycles) are known. Specific parameter settings of the three considered algorithms (PSO, GA and SA) are described in Table 1.1. The parameters used for the ACO algorithm are as follows:

Number of ants = 5
Weight of pheromone trail $\alpha = 1$
Weight of heuristic information $\beta = 5$
Pheromone evaporation parameter $\rho = 0.8$
Constant for pheromone updating $Q = 10$

Each experiment (for each algorithm) was repeated 10 times with different random seeds. Each trial had a fixed number of $50 * m * n$ iterations (m is the number of the grid nodes, n is the number of the jobs). The makespan values of the best solutions throughout the optimization run were recorded. And the averages and the standard deviations were calculated from the 10 different

trials. The standard deviation indicates the differences in the results during the 10 different trials. In a grid environment, the main emphasis will be to generate the schedules at a minimal amount of time. So the completion time for 10 trials were used as one of the criteria to improve their performance.

We tested a small scale job scheduling problem involving 3 machines and 13 jobs (3,13) and 5 machines and 100 jobs (5,100). Fig. 1.5 illustrates the performance of the four algorithms for (3,13). The results for 10 GA runs were {47, 46, 47, 47.3333, 46, 47, 47, 47, 47.3333, 49}, with an average value of 47.1167. The results of 10 SA runs were {46.5, 46.5, 46, 46,46, 46.6667, 47, 47.3333, 47, 47}with an average value of 46.6. The results of 10 PSO runs were {46, 46, 46, 46, 46.5, 46.5, 46.5, 46, 46.5, 46.6667}, with an average value of 46.2667. The results of 10 ACO runs were {46, 46, 46, 46, 46.5, 46.5, 46.5, 46, 46, 46.5}, with an average value of 46.2667. The optimal result is supposed to be 46. While GA provided the best results twice, SA, PSO, ACO provided the best results three, five and six times respectively. Empirical results are summarized in Table 1.2 for (3,13) and (5,100).

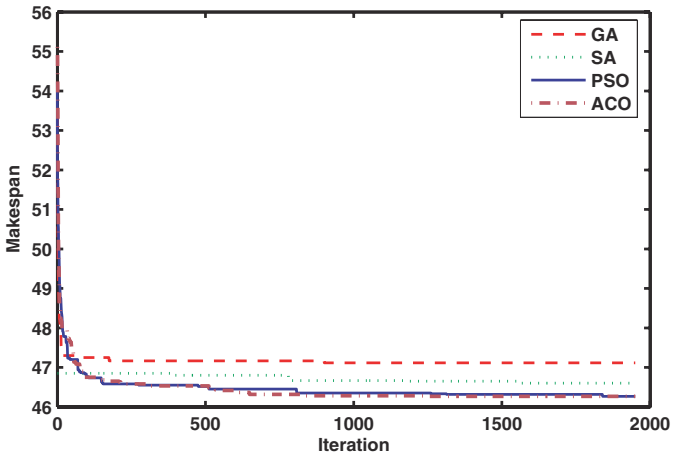

Fig. 1.5. Performance for job scheduling (3,13)

1.4.2 PSO for Data Mining

Data mining and particle swarm optimization may seem that they do not have many properties in common. However, they can be used together to form a method which often leads to the result, even when other methods would be too expensive or difficult to implement. Ujjinn and Bentley [28] provided internet-based recommender system, which employs a particle swarm optimization algorithm to learn personal preferences of users and provide tailored

Table 1.2. Comparing the performance of the considered algorithms.

Algorithm	Item	Instance	
		(3,13)	(5,100)
GA	Average makespan	47.1167	85.7431
	Standard Deviation	±0.7700	±0.6217
	Time	302.9210	2415.9
SA	Average makespan	46.6000	90.7338
	Standard Deviation	±0.4856	±6.3833
	Time	332.5000	6567.8
PSO	Average makespan	46.2667	84.0544
	Standard Deviation	±0.2854	±0.5030
	Time	106.2030	1485.6
ACO	Average makespan	46.2667	88.1575
	Standard Deviation	±0.2854	±0.6423
	Time	340.3750	6758.3

suggestions. Omran et al. [19] used particle swarm to implement image clustering. When compared with K-means, Fuzzy C-means, K-Harmonic means and genetic algorithm approaches, in general, the PSO algorithm produced better results with reference to inter- and intra-cluster distances, while having quantization errors comparable to the other algorithms. Sousa et al. [25] proposed the use of the particle swarm optimizer for data mining. Tested against genetic algorithm and Tree Induction Algorithm (J48), the obtained results indicates that particle swarm optimizer is a suitable and competitive candidate for classification tasks and can be successfully applied to more demanding problem domains. The basic idea of combining particle swarm optimization with data mining is quite simple. To extract this knowledge, a database may be considered as a large search space, and a mining algorithm as a search strategy. PSO makes use of particles moving in an n-dimensional space to search for solutions for an n-variable function (that is fitness function) optimization problem. The datasets are the sample space to search and each attribute is a dimension for the PSO-miner. During the search procedure, each particle is evaluated using the fitness function which is a central instrument in the algorithm. Their values decide the swarm's performance. The fitness function measures the predictive accuracy of the rule for data mining, and it is given by Eq.(1.15):

$$predictive\_accuracy = \frac{|A \wedge C| - 1/2}{|A|} \tag{1.15}$$

where $|A \wedge C|$ is the number of examples that satisfy both the rule antecedent and the consequent, and $|A|$ is the number of cases that satisfy only the rule antecedent. The term $1/2$ is subtracted in the numerator of Eq.(1.15) to penalize rules covering few training examples. PSO usually search the min-

imum for the problem space considered. So we use predictive_accuracy to the power minus one as fitness function in PSO-miner.

Rule pruning is a common technique in data mining. The main goal of rule pruning is to remove irrelevant terms that might have been unduly included in the rules. Rule pruning potentially increases the predictive power of the rule, helping to avoid its over-fitting to the training data. Another motivation for rule pruning is that it improves the simplicity of the rule, since a shorter rule is usually easier to be understood by the user than a longer one. As soon as the current particle completes the construction of its rule, the rule pruning procedure is called. The quality of a rule, denoted by Q, is computed by the formula: $Q = sensitivity \cdot specificity$ [17]. Just after the covering algorithm returns a rule set, another post-processing routine is used: rule set cleaning, where rules that will never be applied are removed from the rule set. The purpose of the validation algorithm is to statistically evaluate the accuracy of the rule set obtained by the covering algorithm. This is done using a method known as tenfold cross validation [29]. Rule set accuracy is evaluated and presented as the percentage of instances in the test set correctly classified. In order to classify a new test case, unseen during training, the discovered rules are applied in the order they were discovered.

The performance of PSO-miner was evaluated using four public-domain data sets from the UCI repository [4]. The used parameters' settings are as following: swarm size=30; $c_1 = c_2 = 2$; maximum position=5; maximum velocity=0.1∼0.5; maximum uncovered cases = 10 and maximum number of iterations=4000. The results are reported in Table 1.3. The algorithm is not only simple than many other methods, but also is a good alternative method for data mining.

Table 1.3. Results of PSO-miner

Data set	Predictive accuracy	Number of rules	Number of terms / Number of rules
Wisconsin breast cancer	92.65 ± 0.61	5.70 ± 0.20	1.63
Dermatology	92.65 ± 2.37	7.40 ± 0.19	2.99
Hepatitis	83.65 ± 3.13	3.30 ± 0.15	1.58
Cleveland heart disease	53.50 ± 0.61	9.20 ± 0.25	1.71

1.5 Ant Colony Optimization

In nature, ants usually wander randomly, and upon finding food return to their nest while laying down pheromone trails. If other ants find such a path (pheromone trail), they are likely not to keep travelling at random, but to instead follow the trail, returning and reinforcing it if they eventually find

food. However, as time passes, the pheromone starts to evaporate. The more time it takes for an ant to travel down the path and back again, the more time the pheromone has to evaporate (and the path to become less prominent). A shorter path, in comparison will be visited by more ants (can be described as a loop of positive feedback) and thus the pheromone density remains high for a longer time.

ACO is implemented as a team of intelligent agents which simulate the ants behavior, walking around the graph representing the problem to solve using mechanisms of cooperation and adaptation. ACO algorithm requires to define the following [5, 10]:

- The problem needs to be represented appropriately, which would allow the ants to incrementally update the solutions through the use of a probabilistic transition rules, based on the amount of pheromone in the trail and other problem specific knowledge. It is also important to enforce a strategy to construct only valid solutions corresponding to the problem definition.
- A problem-dependent heuristic function η that measures the quality of components that can be added to the current partial solution.
- A rule set for pheromone updating, which specifies how to modify the pheromone value τ.
- A probabilistic transition rule based on the value of the heuristic function η and the pheromone value τ that is used to iteratively construct a solution.

ACO was first introduced using the Travelling Salesman Problem (TSP). Starting from its start node, an ant iteratively moves from one node to another. When being at a node, an ant chooses to go to a unvisited node at time t with a probability given by

$$p_{i,j}^k(t) = \frac{[\tau_{i,j}(t)]^\alpha [\eta_{i,j}(t)]^\beta}{\sum_{l \in N_i^k} [\tau_{i,j}(t)]^\alpha [\eta_{i,j}(t)]^\beta} \qquad j \in N_i^k \tag{1.16}$$

where N_i^k is the feasible neighborhood of the ant_k, that is, the set of cities which ant_k has not yet visited; $\tau_{i,j}(t)$ is the pheromone value on the edge (i,j) at the time t, α is the weight of pheromone; $\eta_{i,j}(t)$ is a priori available heuristic information on the edge (i,j) at the time t, β is the weight of heuristic information. Two parameters α and β determine the relative influence of pheromone trail and heuristic information. $\tau_{i,j}(t)$ is determined by

$$\tau_{i,j}(t) = \rho \tau_{i,j}(t-1) + \sum_{k=1}^{n} \Delta\tau_{i,j}^k(t) \qquad \forall (i,j) \tag{1.17}$$

$$\Delta\tau_{i,j}^k(t) = \begin{cases} \frac{Q}{L_k(t)} & \text{if the edge } (i,j) \text{ chosen by the } ant_k \\ 0 & \text{otherwise} \end{cases} \tag{1.18}$$

where ρ is the pheromone trail evaporation rate ($0 < \rho < 1$), n is the number of ants, Q is a constant for pheromone updating.

More recent work has seen the application of ACO to other problems [12, 26]. A generalized version of the pseudo-code for the ACO algorithm with reference to the TSP is illustrated in Algorithm 2.

Algorithm 2 Ant Colony Optimization Algorithm

01. Initialize the number of ants n, and other parameters.
02. While (the end criterion is not met) do
03. $t = t + 1$;
04. For k= 1 to n
05. ant_k is positioned on a starting node;
06. For m= 2 to $problem\_size$
07. Choose the state to move into
08. according to the probabilistic transition rules;
09. Append the chosen move into $tabu_k(t)$ for the ant_k;
10. Next m
11. Compute the length $L_k(t)$ of the tour $T_k(t)$ chosen by the ant_k;
12. Compute $\Delta\tau_{i,j}(t)$ for every edge (i, j) in $T_k(t)$ according to Eq.(1.18);
13. Next k
14. Update the trail pheromone intensity for every edge (i, j) according to Eq.(1.17);
15. Compare and update the best solution;
16. End While.

1.6 Ant Colony Algorithms for Optimization Problems

1.6.1 Travelling Salesman Problem (TSP)

Given a collection of cities and the cost of travel between each pair of them, the travelling salesman problem is to find the cheapest way of visiting all of the cities and returning to the starting point. It is assumed that the travel costs are symmetric in the sense that travelling from city X to city Y costs just as much as travelling from Y to X. The parameter settings used for ACO algorithm are as follows:

Number of ants = 5
Maximum number of iterations = 1000
$\alpha = 2$
$\beta = 2$
$\rho = 0.9$
$Q = 10$

A TSP with 20 cities (Table 1.4) is used to illustrate the ACO algorithm.

The best route obtained is depicted as $1 \rightarrow 14 \rightarrow 11 \rightarrow 4 \rightarrow 8 \rightarrow 10 \rightarrow 15 \rightarrow 19 \rightarrow 7 \rightarrow 18 \rightarrow 16 \rightarrow 5 \rightarrow 13 \rightarrow 20 \rightarrow 6 \rightarrow 17 \rightarrow 9 \rightarrow 2 \rightarrow 12 \rightarrow 3 \rightarrow 1$, and is illustrated in Fig. 1.6 with a cost of 24.5222. The search result for a TSP for 198 cities is illustrated in Figure 1.7 with a total cost of 19961.3045.

Table 1.4. A TSP (20 cities)

Cities	1	2	3	4	5	6	7	8	9	10
x	5.2940	4.2860	4.7190	4.1850	0.9150	4.7710	1.5240	3.4470	3.7180	2.6490
y	1.5580	3.6220	2.7740	2.2300	3.8210	6.0410	2.8710	2.1110	3.6650	2.5560
Cities	11	12	13	14	15	16	17	18	19	20
x	4.3990	4.6600	1.2320	5.0360	2.7100	1.0720	5.8550	0.1940	1.7620	2.6820
y	1.1940	2.9490	6.4400	0.2440	3.1400	3.4540	6.2030	1.8620	2.6930	6.0970

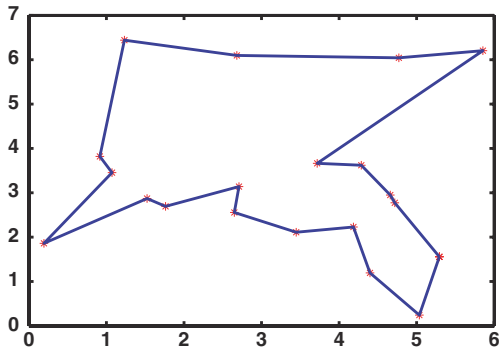

Fig. 1.6. An ACO solution for the TSP (20 cities)

1.6.2 Quadratic Assignment Problem (QAP)

Quadratic assignment problems model many applications in diverse areas such as operations research, parallel and distributed computing, and combinatorial data analysis. There are a set of n facilities and a set of n locations. For each pair of locations a distance is specified and for each pair of facilities a weight or flow is specified (e.g., the amount of supplies transported between the two facilities). The problem is to assign all facilities to different locations with the goal of minimizing the sum of the distances multiplied by the corresponding

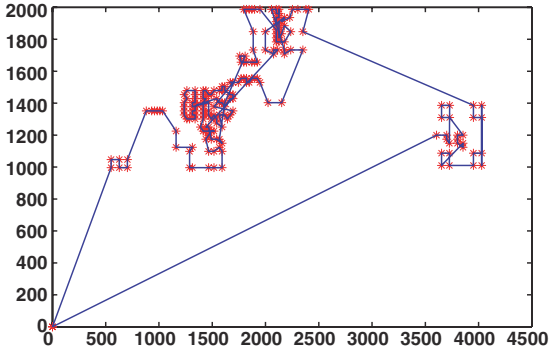

Fig. 1.7. An ACO solution for the TSP (198 cities)

flows. A QAP is used to demonstrate the validity of ACO and its distance/flow matrix for $9 * 9$ assignment is illustrated below:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} * & 1 & 2 & 3 & 1 & 2 & 3 & 4 & 5 \\ 5 & * & 1 & 2 & 2 & 1 & 2 & 3 & 4 \\ 2 & 3 & * & 1 & 3 & 2 & 1 & 2 & 3 \\ 4 & 0 & 0 & * & 4 & 3 & 2 & 1 & 2 \\ 1 & 2 & 0 & 5 & * & 1 & 2 & 3 & 2 \\ 0 & 2 & 0 & 2 & 10 & * & 1 & 2 & 1 \\ 0 & 2 & 0 & 2 & 0 & 5 & * & 1 & 2 \\ 6 & 0 & 5 & 10 & 0 & 1 & 10 & * & 1 \\ 0 & 4 & 0 & 2 & 5 & 0 & 3 & 8 & * \end{pmatrix} \end{array}$$

The parameter settings used for ACO algorithm are as follows:

Number of ants = 5
Maximum number of iterations = 100,000
$\alpha = 1$
$\beta = 5$
$\rho = 0.8$
$Q = 10$

Using ACO, the cheapest cost obtained = 144 and iteration time = 22445. Assignment results are depicted below:

Dept 1 → Site 5; Dept 2 → Site 2; Dept 3 → Site 1; Dept 4 → Site 9; Dept 5 → Site 4; Dept 6 → Site 8; Dept 7 → Site 7; Dept 8 → Site 6; Dept 9 → Site 3.

1.7 Ant Colony Algorithms for Data Mining

The study of ant colonies behavior and their self-organizing capabilities is of interest to knowledge retrieval/management and decision support systems sciences, because it provides models of distributed adaptive organization, which are useful to solve difficult classification, clustering and distributed control problems.

Ant colony based clustering algorithms have been first introduced by Deneubourg et al. [8] by mimicking different types of naturally-occurring emergent phenomena. Ants gather items to form heaps (clustering of dead corpses or cemeteries) observed in the species of *Pheidole Pallidula* and *Lasius Niger*. If sufficiently large parts of corpses are randomly distributed in space, the workers form cemetery clusters within a few hours, following a behavior similar to segregation. If the experimental arena is not sufficiently large, or if it contains spatial heterogeneities, the clusters will be formed along the edges of the arena or, more generally, following the heterogeneities. The basic mechanism underlying this type of aggregation phenomenon is an attraction between dead items mediated by the ant workers: small clusters of items grow by attracting workers to deposit more items. It is this positive and auto-catalytic feedback that leads to the formation of larger and larger clusters.

A sorting approach could be also formulated by mimicking ants that discriminate between different kinds of items and spatially arrange them according to their properties. This is observed in the *Leptothorax unifasciatus* species where larvae are arranged according to their size.

The general idea for data clustering is that isolated items should be picked up and dropped at some other location where more items of that type are present. Ramos et al. [23] proposed *ACLUSTER* algorithm to follow real ant-like behaviors as much as possible. In that sense, bio-inspired spatial transition probabilities are incorporated into the system, avoiding randomly moving agents, which encourage the distributed algorithm to explore regions manifestly without interest. The strategy allows guiding ants to find clusters of objects in an adaptive way.

In order to model the behavior of ants associated with different tasks (dropping and picking up objects), the use of combinations of different response thresholds was proposed. There are two major factors that should influence any local action taken by the ant-like agent: the number of objects in its neighborhood, and their similarity. Lumer and Faieta [18] used an average similarity, mixing distances between objects with their number, incorporating it simultaneously into a response threshold function like the algorithm proposed by Deneubourg et al. [8]. *ACLUSTER* [23] uses combinations of two independent response threshold functions, each associated with a different environmental factor depending on the number of objects in the area, and their similarity. Reader may consult [23] for the technical details of *ACLUSTER*.

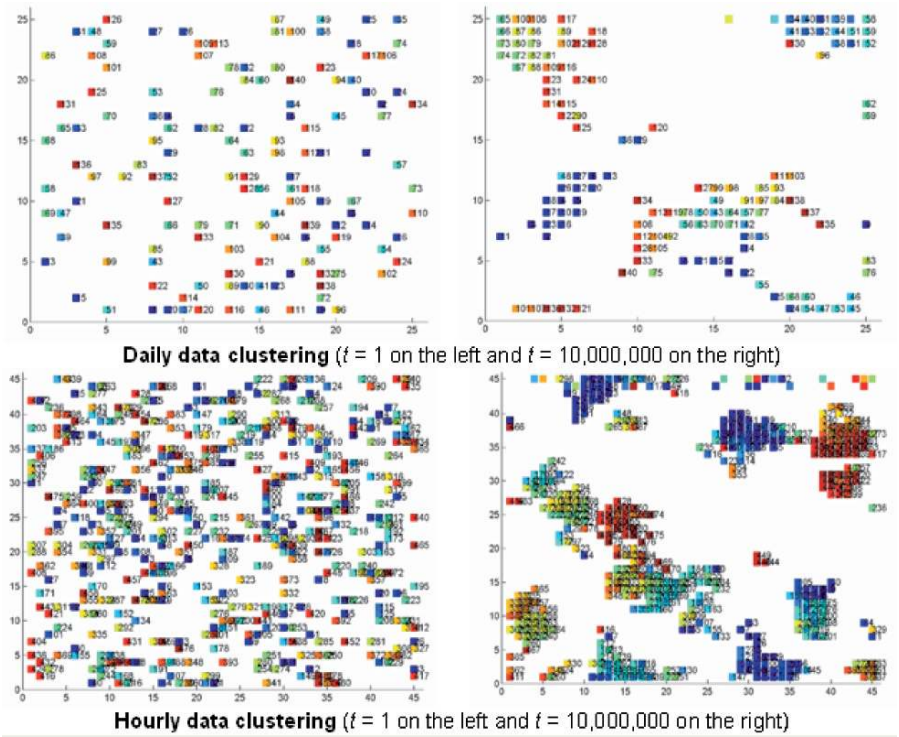

Fig. 1.8. Clustering of Web server visitors using ant colony algorithm (adapted from [3])

1.7.1 Web Usage Mining

Web usage mining has become very critical for effective Web site management, creating adaptive Web sites, business and support services, personalization, network traffic flow analysis etc. [3]. Accurate Web usage information could help to attract new customers, retain current customers, improve cross marketing/sales, effectiveness of promotional campaigns, track leaving customers and find the most effective logical structure for their Web space. User profiles could be built by combining users' navigation paths with other data features, such as page viewing time, hyperlink structure, and page content.

Abraham and Ramos [3] used an ant colony clustering algorithm to discover Web usage patterns (data clusters). The task is to cluster similar visitors accessing the web server based on geographical location, type of information requested, time of access and so on. Web log data of a University server from January 01, 2002 to July 0, 2002 was used in the experiments. The log data was categorized into daily and hourly and for each data set the *ACLUSTER* was run twice for 10,00,000 iterations. A 2D classification space is used which is non-parametric and toroidal.

Experiment results for the daily and hourly Web traffic data are illustrated in Fig. 1.8. Fig. 1.8, at the top, represent the spatial distribution of daily Web traffic data on a 25×25 non-parametric toroidal grid. At t=1, data items are randomly allocated and 14 ants were deployed and as time evolved, several homogenous clusters emerged. Figure 1.8, at the bottom, represent the spatial distribution of hourly Web traffic data on a 45×45 non-parametric toroidal grid. At t=1, data items are randomly allocated and 48 ants were deployed and as time evolved, several homogenous clusters emerged. Reader may consult [3] for detailed results of the different clustering methods.

Clustering results clearly show that ant colony clustering performs well when compared to other clustering methods namely self-organizing maps and evolutionary-fuzzy clustering approach [1].

1.8 Summary

This chapter introduced the theoretical foundations of swarm intelligence with a focus on the implementation and illustration of particle swarm optimization and ant colony optimization algorithms. We provided the design and implementation methods for some applications involving function optimization problems, real world applications and data mining. Results were analyzed, discussed and their potentials were illustrated.

Acknowledgements

First author was supported by the International Joint Research Grant of the IITA (Institute of Information Technology Assessment) foreign professor invitation program of the MIC (Ministry of Information and Communication), South Korea.

References

1. Abraham A (2003) Business intelligence from web usage mining, Journal of Information and Knowledge Management (JIKM), World Scientific Publishing Co., Singapore, 2(4)375-390.
2. Abraham A, Buyya R and Nath B (2000) Nature's heuristics for scheduling jobs on computational grids. Proceedings of the 8th IEEE International Conference on Advanced Computing and Communications, 45-52.
3. Abraham A and Ramos V (2003) Web usage mining using artificial ant colony clustering and genetic programming. Proceesings of IEEE Congress on Evolutionary Computation, Australia, 1384-1391.
4. Blake C, Keogh E and Merz C J (2003) UCI repository of machine learning databases. http://ww.ic.uci.edu/ mlearn/MLRepository.htm.

5. Bonabeau E, Dorigo M and Theraulaz G (1999) Swarm Intelligence: From Natural to Artificial Systems. New York, NY: Oxford University Press.
6. Cantu-Paz E (2000) Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic publishers.
7. Clerc M and Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation, 6(1):58-73.
8. Deneubourg J-L, Goss S, Franks N, at el. (1991) The dynamics of collective sorting: Robot-like ants and ant-like robots. Proceedings of the First International Conference on Simulation of Adaptive Behaviour: From Animals to Animats, Cambridge, MA: MIT Press, 1, 356-365.
9. Dorigo M, Maniezzo V and Colorni A (1996). Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-41.
10. Dorigo M and Stützle T (2004), Ant Colony Optimization, MIT Press, 2004.
11. Eberhart R C and Shi Y (2002) Comparing inertia weights and constriction factors in particle swarm optimization. Proceedings of IEEE International Congress on Evolutionary Computation, 84-88.
12. Gambardella L M and Dorigo M (1995) Ant-Q: A reinforcement learning approach to the traveling salesman problem. Proceedings of the 11th International Conference on Machine Learning, 252-260.
13. Goldberg D E (1989) Genetic Algorithms in search, optimization, and machine learning. Addison-Wesley Publishing Corporation, Inc.
14. Kennedy J and Eberhart R (2001) Swarm intelligence. Morgan Kaufmann Publishers, Inc., San Francisco, CA.
15. Kennedy J and Mendes R (2002) Population structure and particle swarm performance. Proceeding of IEEE conference on Evolutionary Computation, 1671-1676.
16. Liu H and Abraham A (2005) Fuzzy Turbulent Particle Swarm Optimization. Proceeding of the 5th International Conference on Hybrid Intelligent Systems, Brazil, IEEE CS Press, USA.
17. Lopes H S, Coutinho M S and Lima W C (1998) An evolutionary approach to simulate cognitive feedback learning in medical domain. Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives, World Scientific, 193-207.
18. Lumer E D and Faieta B (1994) Diversity and Adaptation in Populations of Clustering Ants. Cli D, Husbands P, Meyer J and Wilson S (Eds.), Proceedings of the Third International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 3, Cambridge, MA: MIT Press, 501-508.
19. Omran M, Engelbrecht P A and Salman A (2005) Particle swarm optimization for image clustering. International Journal of Pattern Recognition and Artificial Intelligence, 19(3):297-321.
20. Orosz J E and Jacobson S H (2002) Analysis of static simulated annealing algorithms. Journal of Optimzation theory and Applications, 115(1):165-182.
21. Pang W, Wang K P, Zhou C G, at el. (2004) Fuzzy discrete particle swarm optimization for solving traveling salesman problem. Proceedings of the 4th International Conference on Computer and Information Technology, IEEE CS Press.
22. Parsopoulos K E and Vrahatis M N (2004) On the computation of all global minimizers through particle swarm optimization. IEEE Transactions on Evolutionary Computation, 8(3):211-224.

23. Ramos V, Muge F, Pina P (2002) Self-organized data and image retrieval as a consequence of inter-dynamic synergistic relationships in artificial ant colonies. Soft Computing Systems - Design, Management and Applications, Proceedings of the 2nd International Conference on Hybrid Intelligent Systems, IOS Press, 500-509.
24. Shi Y H and Eberhart R C (2001) Fuzzy adaptive particle swarm optimization. Proceedings of IEEE International Conference on Evolutionary Computation, 101-106.
25. Sousa T, Silva A, Neves A (2004) Particle swarm based data mining algorithms for classification tasks. Parallel Computing, 30:767-783.
26. Stützle T and Hoo H H (2000) MAX-MIN ant system. Future Generation Computer Systems, 16:889-914.
27. Triki E, Collette Y and Siarry P (2005) A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. European Journal of Operational Research, 166:77-92.
28. Ujjin S and Bentley J P (2003) Particle swarm optimization recommender system. Proceeding of IEEE International conference on Evolutionary Computation, 124-131.
29. Witten Ian H and Frank E (1999) Data mining - Practical Machine Learning Tools and Techniques with Java Implementations. CA: Morgan Kauffmann.