10

Swarm Intelligent Systems

Go to the ant, thou sluggard; consider her ways, and be wise: which having no guide, overseer, or ruler, provideth her meat in the summer and gathereth food in the harvest.

The Holy Bible

After reading this chapter, you will be able to understand the following:

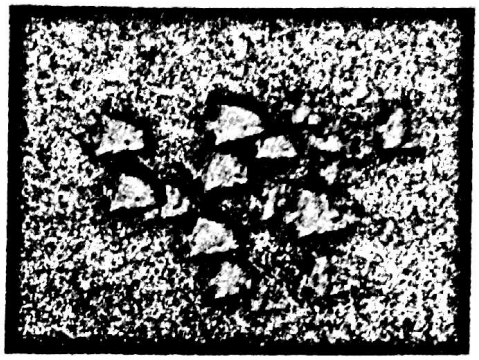
- Swarm intelligence
- Background and development of ant colony intelligent systems
- Basic concepts of the ant colony metaphor—biological and artificial ant colony systems
- Different types of ant colony models and their features
- Engineering applications of ant colony intelligent systems
- Background and development of particle swarm intelligent systems
- Basic concepts and engineering applications of particle swarm intelligent systems

10.1 INTRODUCTION

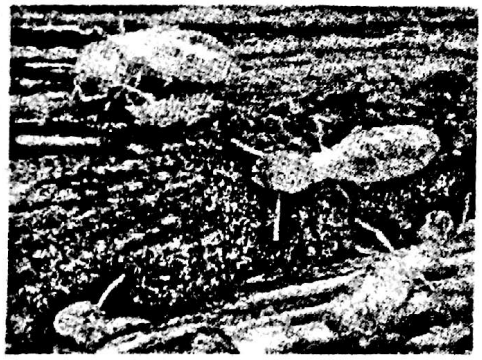
Our civilization has long since recognized the significance of the creativity of human endeavour in every discipline and field. Nature has guided us to watch and learn the intelligent mechanism evolved by it. The marching of ants in an army, the waggle dance of the honeybee, the nest building of the social wasp, birds flocking in high skies, fish schools in deep waters, the foraging activities of microorganisms,

the construction of the termite mound, etc. have puzzled biologists over the years (Fig. 10.1). The last decade has seen an explosion of research in this field variously referred to as collective intelligence, swarm intelligence, and emergent behaviour, during the course of which scientists have unravelled many of the mysteries surrounding social insects.

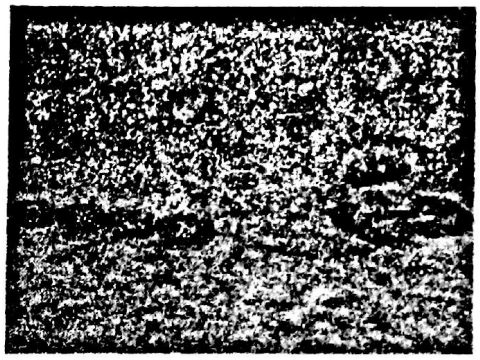
(a) A flock of birds



(b) A school of fish



(c) Termites (*Rhinotermitidae*) feeding on wood (http://tolweb.org)



(d) Foraging *E. coli* bacteria

(e) A swarm of honeybees

(f) A group of *Pheidole nodus* ants carrying a spider

Fig. 10.1 Social living beings

In the recent past, the swarm paradigm has been applied to a broader range of studies, opening up new views of theoretical biology, economics, and philosophy. Many instances of creativity in animals arise from collective behaviour, not from a single individual's actions. The term swarm intelligence is used for the collective

behaviour of a group (swarm) of animals as a single living creature, where *collective intelligence* emerges via grouping and communication, actually resulting in more successful foraging for each individual in the group.

Swarm intelligence is a specialization in the field of self-organizing systems (*adaptation*). When the route of a swarm of ants is blocked, it can be observed that they find another new shortest route to their destination; this shows *robustness*. These agents (ants) can be added or removed without compromising the total system due to its distributed nature. This adaptation is more *reliable*. Single parts may break down without impairing the overall system. These complex systems are convenient to work with because of the *simplicity* of their individual parts. The desired characteristics emerge from the interaction of the various parts, without explicit supervision or a central control system, which is intelligent behaviour. Knowledge is distributed and becomes apparent in the interaction between the agents and the environment (http://www.ai.rug.nl).

The *optimal foraging policy* has been learned from the biological phenomenon of these socially adapting living creatures. *Foraging theory* is based on the assumption that animals search for and obtain nutrients in a way that maximizes their energy intake E per unit time T spent foraging. The maximization of such a function provides nutrient sources for the animal to survive and additional time for other important activities (e.g., fighting, fleeing, mating, reproducing, sleeping, or shelter-building). Shelter-building and mate-finding activities are sometimes similar to foraging. Clearly, foraging is very different for different species. The foraging formulation is only meant to be a model that explains optimal behaviour. Lower life forms can achieve higher forms of foraging intelligence by cooperating in groups (e.g., ants 'learn' where food is located, and the foraging strategy of the *Escherichia coli* bacteria and honeybees) (Passino 2000; 2002).

Ants have been living on the earth for more than 100 million years and can be found almost anywhere on the planet. It is estimated that there are about 20,000 different species of ants. For this reason, they have been called the earth's most successful species. Ants are *social insects*, which means they live in large colonies or groups. Some colonies consist of millions of ants. The eyes of ants are made up of many lenses enabling them to see movement very well. Their antennae are special organs of smell, touch, taste, and hearing. If you watch ants for some time, you will see that they really do communicate with each other, and very effectively too. They communicate by touching each other with their antennae. They also use chemicals called *pheromones* to leave scent trails for other ants to follow. Leafcutter ants (*Atta*), carpenter ants (*Hymenoptera*), weaver ants (*Oecophylla*), army ants (*Eciton*), etc. have their own specialized activities for survival. Leafcutter ants mostly found in the American tropics are the most important herbivores (plant consumers), outranking grazing mammals. In many ecosystems, ants are important dispersers of seeds. They help in maintaining the health of the environment by fertilizing the soil. The raid patterns of ant armies, which contain 200,000 workers raiding in dense, virtually blinding troops of 15 m or wider, are very exciting and

dynamic in nature, exhibiting the same basic structure—a powerful decentralized control system (Bonabeau et al. 1999). From the social life of these ants, it is apparent that a strong co-existence between individuals, adaptation according to the needs of the colony, foraging behaviour, etc. are the qualities that form the basis of this swarm (ant) intelligent system.

Biomimicry of the bacterial foraging of *E. coli* (present in the large intestine of human beings), which was modelled by Prof. K.M. Passino of Ohio State University, is also a good example of swarm intelligent systems for distributed optimization. The biological phenomena such as chemotaxis, swarming, reproduction, elimination, and dispersal events seen in these organisms are mathematically modelled to suit optimization problems (Passino 2000). After finding a nectar source, a honeybee goes back to its hive, gives up its nectar to the hive bees, and starts dancing to inform other bees of the direction and the distance of the food source. If there are two sources at equal distances, the bees exploit the better source first or abandon the poor source, thus selecting better quality food. This intelligent behaviour can be used to find better quality solutions in optimization problems and formulate better decision-making mechanisms.

There are two popular swarm-inspired methods in the computational intelligence area: *ant colony optimization* (ACO) and *particle swarm optimization* (PSO). Invented by an Italian scientist named Marco Dorigo (2004), ACO was inspired by the behaviour of ants and has many successful applications in discrete optimization problems. Particle swarm optimization is a population-based stochastic optimization technique developed by Eberhart and Kennedy in 1995, inspired by the social behaviour of flocks of birds and schools of fish.

In particle swarm intelligent systems (PSIS), bird flocking behaviour is simulated for optimization problems. Consider the following situation. A group of birds are randomly searching for food in an area. Only one particular location of the area being searched contains food. None of the birds knows where the food is. The sole knowledge they have is how far the food is in each iteration. So, the best and most effective strategy to find the food is to follow the bird that is nearest to it. PSIS have learned from this scenario and used it to solve optimization problems.

In the PSIS formulation, each single solution is a 'bird' in the search space and can be treated as a 'particle'. Each particle has a fitness value. The fitness function to be optimized is evaluated for every particle, and has velocities which direct the flying of the particles. The particles fly through the problem space by following the currently optimum particles (http://www.engr.iupui.edu). In particle swarm optimization methods, each particle is initialized with a group of random particles, which are solutions; optima are searched for by updating subsequent generations. In every iteration, each particle is updated by following two 'best' values. The first best value is the best solution or fitness it has achieved so far. The solution corresponding to the best fitness value is stored and called *pbest*. The other best value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle in the population. This best value is a global best and is called

gbest. These best solutions are obtained from a relation maintained by the current particle's velocity and position. However, the intelligent mechanism involved in PSIS that provides the solution to a defined problem is different from the mechanism of ant colony intelligent systems, which will be detailed in the following sections.

10.2 BACKGROUND OF ANT INTELLIGENT SYSTEMS

Marco Dorigo and his colleagues first proposed ant algorithms in the year 1991 as a multi-agent approach to solve difficult combinatorial optimization problems. There is currently a lot of ongoing activity in the scientific community to extend/apply ant-based algorithms to many different discrete optimization problems. The application and experimental validation of these algorithms are being thoroughly researched owing to their capability to provide an almost global optimal solution to a given complex problem structure such as local search, image mapping and compression, database search, etc. Other than the examples enumerated above, there are many unexplored situations where ant algorithms can be successfully applied (Dorigo 1992; Dorigo et al. 1996).

Ant algorithms were first tested and validated on the travelling salesman problem (TSP). The TSP was chosen for several reasons, one of them being that it is a shortest path problem for which the ant colony metaphor can easily be adopted. The main idea is that of having a set of agents, called ants, search in parallel for good solutions and cooperate through the pheromone-mediated indirect method of communication (Bonabeau et al. 1999). Table 10.1 gives a brief history of the development of ant colony systems (ACSs) with respect to different applications. Ant algorithms belong to a class of meta-heuristics, which have a gamut of applications to practical problems faced in business and industrial environments. The various applications described below highlight the fact that ant algorithms can be applied to many of these practical scenarios (Dorigo et al. 1999).

Table 10.1 List of ant colony optimization (ACO) algorithms in chronological order for a particular problem type

Authors	*Year*	*Algorithm*	*Problem type*
Dorigo, Maniezzo, and Colorni	1991	AS	Travelling salesman
Gambardella and Dorigo	1995	Ant-Q	
Dorigo and Gambardella	1996	ACS and ACS-3-opt	
Stützle and Hoos	1997	*MM*AS	
Bullnheimer, Hartl, and Strauss	1997	AS rank	
Maniezzo, Colorni, and Dorigo	1994	AS-QAP	Quadric assignment
Gambardella, Taillard, and Dorigo	1997	HAS-QAP	
Stützle and Hoos	1998	*MM*AS-QAP[a]	
Maniezzo and Colorni	1999	AS-QAP[b]	
Maniezzo	1998	ANTS-QAP	

(contd)

Table 10.1 *(contd)*

Authors	*Year*	*Algorithm*	*Problem type*
Colorni, Dorigo, and Maniezzo	1994	AS-JSP	Job shop scheduling
Bullnheimer, Hartl, and Strauss	1996	AS-VRP	Vehicle routing
Gambardella, Taillard, and Agazzi	1999	HAS-VRP	
Gambardella and Dorigo	1997	HAS-SOP	Sequential ordering
Costa and Hertz	1997	ANTCOL	Graph colouring
Michel and Middendorf	1998	AS-SCS	Shortest common super-sequence
Schoonderwoerd, Holland, Bruten, and Rothkrantz	1996	ABC	Connection-oriented network routing
White, Pagurek, and Oppacher	1998	ASGA	
Di Caro and Dorigo	1998	AntNet-FS	
Bonabeau, Henaux, Gúerin, Snyers, Kuntz, and Theraulaz	1998	ABC-smart ants	
Di Caro and Dorigo	1997	AntNet and AntNet-FA	Connection-less network routing
Subramanian, Druschel, and Chen	1997	Regular ants	
Heusse, Gúerin, Snyers, and Kuntz	1998	CAF	
Van der Put and Rothkrantz	1998	ABC-backward	
Forsyth and Wren	1997	Ant system (AS)	Bus driver scheduling
Stützle	1998	ACO	Flow manufacturing
Bland	1999	AS (TS)	Layout of facilities
Bland	1999	ACO	Space planning
Zhou and Liu	1999	Intelligent ant algorithm	Dynamic routing in telecommunication networks
Monmarché, Venturini, and Slimane	2000	API	Numeric optimization
Tzafestas	2000	Painter ants	Painter ants
Jong and Wiering	2001	Multiple ant colony systems	Bus stop allocation problem
Jayaraman et al.	2001	ACO	Bioreactors optimization
Gravel, Price, and Gagn	2001	ACO	Job scheduling in aluminium foundries

10.3 IMPORTANCE OF THE ANT COLONY PARADIGM

The evolving computational paradigm of ant colony intelligent systems (ACIS) is being used as an intelligent tool to help researchers solve many problems in different areas of science and technology. Scientists nowadays are using the functions of real ant colonies to solve many combinatorial optimization problems in different engineering applications. Traditional, sequential, logic-based digital computing excels in many areas, but has not been very successful for certain types of problems. Features such as positive feedback, distributed computation, and constructive greedy heuristic approaches are some important characteristics that have helped the research community formulate successful implementations of ACIS, which have tremendous potential for the future.

10.4 ANT COLONY SYSTEMS

An artificial ant colony system (AACS) is a random stochastic population-based heuristic algorithm of agents that simulate the natural behaviour of ants, developing mechanisms of cooperation and learning, which enables the exploration of the positive feedback between agents as a search mechanism (Dorigo 1992; Dorigo et al. 1996; Dorigo & Gambardella 1997; Dorigo et al. 1999; Dorigo & Stützle 2000; Meuleau & Dorigo 2000).

10.4.1 Biological Ant Colony Systems

Social insects like ants, bees, wasps, and termites perform their simple tasks themselves, independently of other members of the colony. However, when they act as a community, they are able to solve the complex problems emerging in their daily lives through mutual cooperation. This emergent behaviour of self-organization in a group of social insects is known as *swarm intelligence*, which has four basic ingredients (a) positive feedback, (b) negative feedback (e.g., saturation, exhaustion, competition), (c) amplification of fluctuations (e.g., random walk, errors, random task switching), and (d) multiple interaction (Bonabeau et al. 1999). Swarm intelligent systems are hard to program since the paths to problem solving are not predefined, but emergent in the system itself due to the interactions between individuals, those between individuals and their environment, or the behaviour of the individuals themselves. An important and interesting behaviour of ant colonies is their foraging behaviour and, in particular, ability to find the shortest paths between food sources and their nests. While walking from food sources to their nest and vice versa, ants deposit a chemical substance called pheromone on the ground, forming in this way a *pheromone trail*. The sketch shown in the Fig. 10.2 gives a general idea of the pheromone trail.

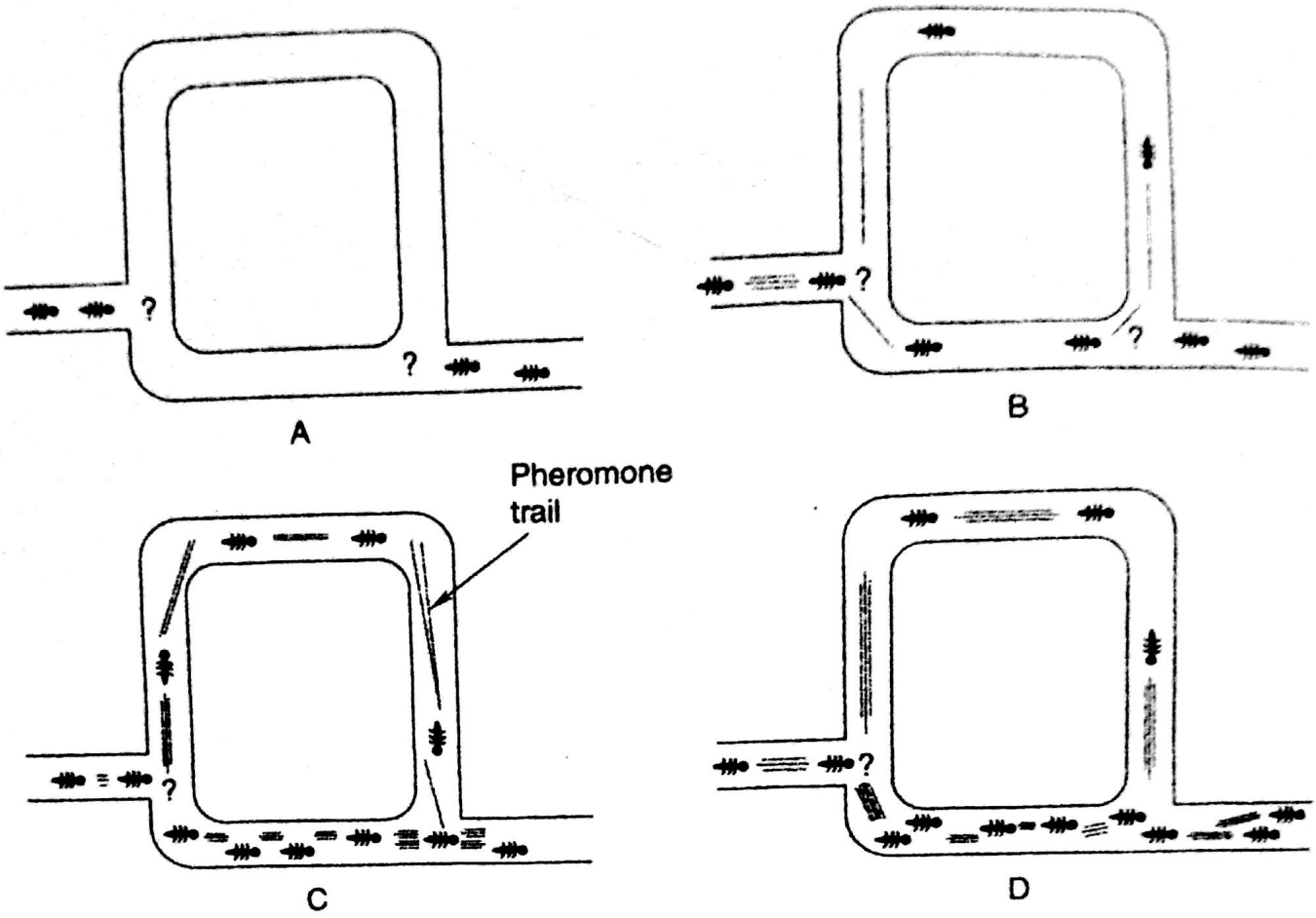


Fig. 10.2 Foraging behaviour of ants moving from their nest (origin) to the food source (destination), taking the shortest possible route through pheromone mediation [stage (a) to stage (d)]

How do real ants find the shortest path? Ants can smell pheromones; while choosing their path, they tend to choose the paths marked by strong pheromone concentrations. The pheromone trail allows ants to find their way back to the food. The emergence of this shortest path selection behaviour can be explained in terms of *autocatalysis* (positive feedback) and differential path length, which uses a simple form of indirect communication, such as mediation through pheromone laying. known as *stigmergy* through the environment, either by physically changing it in some way or by depositing something in the environment (e.g., an ant's laying a pheromone trail on the ground indirectly allows other ants to follow it to find their food source; deposits of soil pellets in a termite colony stimulates all the workers to accumulate more material through a positive feedback mechanism, since the accumulation of the material reinforces the attraction of the deposits through the diffusing pheromone emitted by the pellets). The process whereby an ant is influenced toward a food source by another ant or by a chemical trail is called *recruitment*; recruitment based solely on chemical trails is called *mass recruitment* (Dorigo 1992; Dorigo & Gambardella 1997; Bonabeau et al. 1999; Dorigo & Di Caro 1999; Simon et al. 2003).

10.4.2 Artificial Ant Colony Systems

In AACSs, the use of (a) a colony of cooperating individuals, (b) an artificial pheromone trail for local stigmergetic communication, (c) a sequence of local moves for finding the shortest paths, and (d) a stochastic decision policy using local information and no look-ahead are the same as in real ACSs. However, artificial ants also have some characteristics which do not find counterparts in real ants (Dorigo et al. 1999). These are listed below.

(1) Artificial ants live in a discrete world and their moves consist of transitions from discrete states to discrete states.
(2) Artificial ants have an internal state. This private state contains the memory of the ant's past actions.
(3) Artificial ants deposit a particular amount of pheromone, which is a function of the quality of the solution found.
(4) An artificial ant's timing in pheromone laying is problem-dependent and often does not reflect a real ant's behaviour. For example, in many cases, artificial ants update pheromone trails only after having generated a solution.
(5) To improve overall system efficiency, ant algorithms can be enriched with extra capabilities such as look-ahead, local optimization, backtracking, elistic approach, ranking-based approach, etc., which cannot be found in real ants.

10.5 DEVELOPMENT OF THE ANT COLONY SYSTEM

The ant system (AS) was the first example of an ant colony optimization (ACO) algorithm and, in fact, originally a set of three algorithms called *ant-cycle*, *ant-density*, and *ant-quantity*. These three algorithms were proposed in Dorigo's doctoral dissertation (Dorigo 1992). While in ant-density and ant-quantity, ants can update the pheromone trail directly after a move from one node to an adjacent one, in ant-cycle the pheromone update was carried out only after all the ants had constructed their tours and the amount of pheromone deposited by each ant was set to a function denoting the tour quality. Since ant-cycle performed better than the other two variants, it was later simply called ant system, while the other two algorithms were no longer studied.

The major merit of the AS, whose computational results were promising but not competitive enough as compared to other more established approaches, was to stimulate a number of researchers to develop extensions and improvements of its basic ideas so as to produce more performing, and often state-of-the-art, algorithms. It was following the successes of this collective undertaking that Dorigo and Di Caro recently made the synthesis effort that took the definition of the ACO meta-heuristic. In other words, the ACO meta-heuristic was defined *a posteriori* with the goal of providing a common characterization of this new class of algorithms

and a reference framework for the design of new instances of ACO algorithms (Dorigo et al. 1996; Dorigo & Stützle 2000).

10.6 APPLICATIONS OF ANT COLONY INTELLIGENCE

There are now numerous successful implementations of the ACO meta-heuristic applied to a number of different combinatorial optimization problems. Looking at these implementations, it is possible to distinguish between two classes of applications: static combinatorial optimization problems and dynamic ones. Also, classification-rule-based problems as well as problems where decision making is very important are being tried out and are showing significant improvement. Researchers are trying to use ACO in hybrid models, which are a combination of different intelligent techniques.

10.6.1 Static Combinatorial Optimization Problems

Static problems are those in which the characteristics of the problem are given once and for all when the problem is defined, and do not change while the problem is being solved. The application of the ACO meta-heuristic to a static combinatorial optimization problem is more or less straightforward, once a mapping of the problem is defined, which allows the incremental construction of a solution; a neighbourhood structure and a stochastic state transition rule are locally used to direct the constructive procedure. A strictly implementation-dependent aspect of the ACO meta-heuristic regards the timing of pheromone updates. In ACO algorithms for static combinatorial optimization, the way ants update pheromone trails changes across algorithms: any combination of online step-by-step pheromone updates and online delayed pheromone updates is possible.

A typical example of such problems is the classic travelling salesman problem in which city locations and their relative distances are a part of the problem definition and do not change at run-time. Other applications such as quadratic assignment, job shop scheduling, vehicle routing, sequential ordering, graph colouring, and shortest common super-sequence are some combinatorial optimization problems that have been successfully implemented (Dorigo & Stützle 2000).

10.6.2 Dynamic Combinatorial Optimization Problems

Dynamic combinatorial optimization problems are defined as functions of some quantities whose values are set by the dynamics of an underlying system. The problem changes therefore at run-time and the optimization algorithm must be capable of adapting online to the changing environment. A paradigmatic example is the problem of network routing. Research on the applications of ACO algorithms to dynamic combinatorial optimization problems has focused mainly on communication networks. This is primarily due to the fact that network optimization problems have characteristics like inbuilt information and computation distribution,

non-stationary stochastic dynamics, and asynchronous evolution of the network status, which well match those of the ACO meta-heuristic. In particular, the ACO approach has been applied to routing problems such as connection-oriented network routing and connection-less network routing. Routing is one of the most critical components of network control and concerns the network-wide distributed activity of building and using routing tables to direct data traffic (Dorigo & Stützle 2000).

10.7 THE WORKING OF ANT COLONY SYSTEMS

Essentially, an ACS algorithm performs a loop, applying two basic procedures:

- specifying how ants construct or modify a solution for the problem in hand, and
- updating the pheromone trail.

The construction or modification of a solution is performed in a probabilistic way. The probability of adding a new term to the solution under construction is, in turn, a function of a problem-dependent heuristic and the amount of pheromone previously deposited in this trail. The pheromone trails are updated considering the evaporation rate and the quality of the current solution.

10.7.1 Probabilistic Transition Rule

In a simple ACO algorithm, the main task of each artificial ant, similar to their natural counterparts, is to find a shortest path between a pair of nodes on a graph on which the problem representation is suitably mapped. Let $G = (N, A)$ be a connected graph with $n = |N|$ nodes. The simple ant colony optimization (S-ACO) algorithm can be used to find the solution to a shortest path problem defined on the graph G, where a solution is a path on the graph connecting a source node s to a destination node d shown in Fig. 10.3, and the path length given by the number of loops in the path to each arc (i, j) of the graph is associated with a variable τ_{ij} called an *artificial pheromone trail*. At the beginning of the search process, a small amount of pheromone τ_0 is assigned to all the arcs. Pheromone trails are read and written by ants. The amount (intensity) of a pheromone trail is proportional to the utility, as estimated by the ants, of that arc to build good solutions. Each ant applies a step-by-step constructive decision policy to build the problem's solution. At each node, local information maintained in the node itself and/or in its outgoing arcs is used in a stochastic way to decide the next node to move to (Bonabeau et al. 1999).

The decision rules of an ant k located in node i use the pheromone trails τ_{ij} to compute the probability with which it should choose node $j \in N_i$ as the next node to move to, where N_i is the set of one-step neighbours of node i:

$$p_{ij}^k = \begin{cases} \tau_{ij} / \sum_{j \in N_i} \tau_{ij} & \text{if } j \in N_i \\ 0 & \text{if } j \notin N_i \end{cases} \qquad (10.1)$$

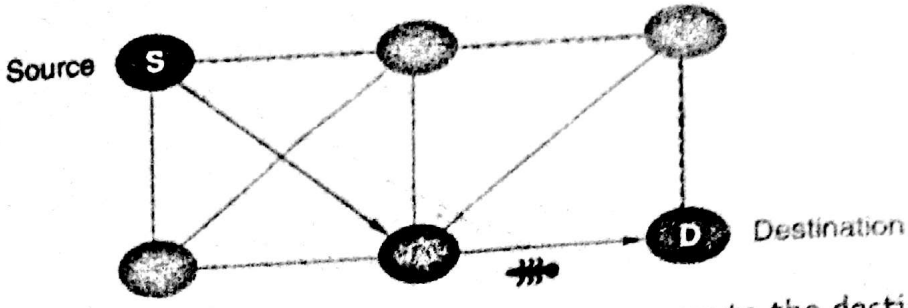


Fig. 10.3 Building of solutions by an ant from the source to the destination node

10.7.2 Pheromone Updating

While building a solution, ants deposit pheromone information on the arcs they use. In S-ACO, ants deposit a constant amount $\Delta\tau$ of pheromone. Consider an ant that at time t moves from node i to node j. It will change the pheromone value τ_{ij} as follows:

$$\tau_{ij}(t) \leftarrow \tau_{ij}(t) + \Delta\tau \tag{10.2}$$

Using this rule, which simulates real ants' pheromone deposits on arc (i, j), an ant using the arc connecting node i to node j increases the probability that other ants will use the same arc in the future. As in the case of real ants, autocatalysis and differential path length are at work to favour the emergence of short paths. To avoid a quick convergence of all the ants towards a sub-optimal path, an exploration mechanism is added: similar to real pheromone trails, artificial pheromone trails evaporate. In this way, the pheromone intensity decreases automatically, favouring the exploration of different arcs during the whole search process. The evaporation is carried out in a simple way, decreasing pheromone trails exponentially, $\tau = (1 - \rho)\tau$, $\rho \in (0, 1)$, in each iteration of the algorithm. The way the pheromone trail is updated can be classified mainly into three types as detailed below (Bonabeau et al. 1999).

Online step-by-step pheromone update When moving from node i to neighbouring node j, the ant can update the pheromone trail τ_{ij} on the arc (i, j).

Online delayed pheromone update Once a solution is built, the ant can retrace the same path backward and update the pheromone trails on the traversed arcs.

Off-line pheromone update Pheromone updates performed using the global information available are called off-line pheromone updates. Here, by observing the ant's behaviour and with the help of global information like the shortest path the ant has taken or on the basis of the solution generated by the ants, additional pheromone is deposited on the arcs traversed by the ants, enabling the ant search process to proceed from a non-local perspective.

10.7.3 Assessment of Solutions

In ACO algorithms, the solutions generated by the ants provide feedback in order to direct the search of the future ants entering the system. This is done using two

mechanisms, i.e., explicit and implicit solution evaluation. In explicit solution evaluation, some measure of the quality of the solution generated is used to decide how much pheromone should be deposited by an ant. In implicit solution evaluation, the ants exploit the differential path length (DPL) effect of real ant foraging behaviour; i.e., if an ant chooses a shorter path, it is the first to deposit pheromone on it and bias the search of approaching ants.

In geographically distributed problems like network problems, implicit solution evaluation based on the DPL effect can perform a vital role, where explicit solution is switched off by setting the amount of pheromone deposited by ants to a constant value independent of the cost of the path built by them. Thus, it is possible to find good solutions to network problems by exploiting the DPL effect. It can also be proved that coupling explicit and implicit solution evaluation by making the amount of pheromone deposited proportional to the cost of the solution generated improves the performance. The DPL effect can be efficiently and naturally exploited only in geographically distributed network problems without incurring any additional computational cost due to the distributed nature of nodes in routing problems. This is due to the decentralized nature of the system and the inherently asynchronous nature of the dynamics of a real network. Asynchronous ants are not used in combinatorial optimization problems due to the computational inefficiencies introduced by them and the fact that asynchronous ants can outweigh the gains of the DPL effect (Dorigo & Gambardella 1997; Dorigo et al. 1999; Dorigo & Di Caro 1999; Dorigo and Thomas Stützle 2000; Meuleau & Dorigo 2000).

10.7.4 Types of Ant Colony Models

Different types of ant colony models (ACMs) have evolved over the last few years. Some of these are listed here.

(1) Ant colony optimization meta-heuristic model for discrete optimization problems
(2) Ant system model for the travelling salesman problem
(3) Ant-density
(4) Ant-quantity
(5) Ant-cycle
(6) AS-JSP (for the job shop scheduling problem)
(7) Ant colony system models
(8) Ant-Q
(9) Ant colony system-3-opt
(10) AS-QAP (for the quadratic assignment problem)
(11) Max-min ACS (using the trail smooth mechanism)
(12) AS ranking model
(13) Ant net (for best effort routing in connection-less networks)
(14) Ant-based control (ABC; for connection-oriented network routing)

(15) Ant net-FA (improved version of ant net using *flying ants* for high-priority queues)
(16) Ant net-FS (adaptive *fair share* in multipath high-speed connection-oriented networks)
(17) Ant miner for rule discovery in databases

10.7.4.1 ACO algorithms for the travelling salesman problem

Different types of ACO algorithms have developed for the TSP. The TSP is considered due to its suitability and flexibility for the implementation of the ant search mechanism easily without too many technical difficulties for shortest path problems. In addition, TSP is a hard combinatorial optimization problem; it is more appropriate for comparing and validating solutions when results are already available and is solved using different algorithms for many sets of standard test problems (http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/tsp/). In the following section, some important ant algorithms, their choice in selecting a path, and the way they update the pheromone trail is discussed with respect to the TSP.

Background The TSP is that of a salesman who wants to find the shortest possible trip starting from his home city through a given set of customer cities and returning to his home city. It can be represented by a complete weighted graph $G = (N, A)$, where N is the set of nodes representing the cities and A is the set of arcs fully connecting the nodes N. Each arc is assigned a value d_{ij}, which is the length of arc $(i, j) \in A$, that is, the distance between cities i and j, with $(i, j) \in N$. The TSP is the problem of finding a minimal length Hamiltonian circuit of the graph. A Hamiltonian circuit is a closed tour visiting each of the $n = |N|$ nodes of G exactly once.

For symmetric TSPs, the distances between the cities are independent of the direction in which the arcs are traversed, that is, $d_{ij} = d_{ji}$ for all pairs of nodes. In the more general asymmetric TSP (ATSP), there exists at least one pair of nodes (i, j) for which $d_{ij} = d_{ji}$. In the case of symmetric TSPs, Euclidean TSP instances are used, in which the cities are points in the Euclidean space and the intercity distances are available. ACO algorithms for the TSP are computed using the Euclidean norm (i.e., $d_{ij} = [(x_i - x_j)^2 - (y_i - y_j)^2]^{1/2}$). The ATSP is more difficult to solve than the TSP; in fact while the symmetric TSP can be solved optimally even on graphs with several thousand nodes, ATSP instances, and particularly ATSP instances where the distance matrix is almost symmetric, can be solved optimally only on graphs with a few dozen nodes. All the TSP instances discussed in this chapter have been taken from the TSPLIB Benchmark library, which contains a large collection of such instances; some of these have been used in many other studies, while some stem from practical applications of the TSP. In all these ACO models, the transition rule and the way the pheromone trail updates vary (Dorigo & Gambardella 1997; Dorigo & Stützle 1999).

Action choice transition rule The transition probability for the kth ant from one state i to the next state j for the ant system (AS) model is given by

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{k \in \text{allowed}_k} [\tau_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}} & \text{if } j \in \text{allowed}_k \\ 0 & \text{otherwise} \end{cases} \tag{10.3}$$

where τ_{ij} is the trail intensity on edge (i, j), $\eta_{ij} = (1/d_{ij})$ is the heuristic function, d_{ij} is the Euclidean distance calculated for that particular stage, α is the relative importance of the trail $(\alpha \geq 0)$, β is the relative importance of the visibility $(\beta \geq 0)$, and allowed_k is the set of available states (from the ith state to the jth state) from which the kth ant can choose. The probability transition rule for the ACS is modified to allow explicitly for exploration. An ant k in a state i chooses the next state j to move to according to the following rule:

$$j = \begin{cases} \arg\max_{u \in J_i^k} \{[\tau_{iu}(t)][\eta_{iu}]^{\beta}\} & \text{if } q \leq q_0 \\ J & \text{if } q > q_0 \end{cases} \tag{10.4}$$

where q is a random variable uniformly distributed over [0, 1], q_0 is a tunable parameter $(0 \leq q_0 \leq 1)$, and $J \in J_i^k$ is a state that is randomly selected according to the probability

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)][\eta_{ij}]^{\beta}}{\sum_{l \in J_i^k} [\tau_{il}(t)][\eta_{il}]^{\beta}} \end{cases} \tag{10.5}$$

$q \leq q_0$ corresponds to the exploitation of the knowledge available about the problem, that is, the heuristic knowledge of the cost between states and the learned knowledge memorized in the form of pheromone trails, whereas $q > q_0$ favours more exploration. Cutting exploration by tuning q_0 allows the system's activity to be concentrated on the best solutions instead of letting it explore constantly. Here l denotes an allowable state (Bonabeau et al. 1999).

Pheromone trail update rule After choosing a path with the help of the action choice rule, an ant has to indicate its movement and the quality of the food source (solution) obtained so far to its fellow ants. This is achieved with the help of pheromone-mediated indirect communication (stigmergy), which creates a feedback mechanism as explained earlier. This helps fellow ants to decide whether to take the present route or select another one. The pheromone-updating method is different for different types of ant colony models, as described in the following.

Ant-quantity model In this model, ants locally update the pheromone trail directly after a moving from one state to the adjacent one:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij}(t, t+1) \tag{10.6}$$

where

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/d_{ij} & \text{if the } k\text{th ant uses edge } (i, j) \text{ in its tour} \\ & \text{(between time } t \text{ and } t + 1) \\ 0 & \text{otherwise} \end{cases} \tag{10.7}$$

m is the total number of ants considered, and Q is a constant related to the quantity of the pheromone in the trail.

Ant-density model In the ant-density model, an ant going from i to j leaves a quantity Q of the pheromone on edge (i, j) every time it goes from i to j. Hence, the ant-density model is represented as

$$\tau_{ij}(t + 1) = \rho\tau_{ij}(t) + \Delta\tau_{ij}(t, t + 1) \tag{10.8}$$

where

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} Q & \text{if the } k\text{th ant uses edge } (i, j) \text{ in its tour} \\ & \text{(between time } t \text{ and } t + 1) \\ 0 & \text{otherwise} \end{cases} \tag{10.9}$$

Ant-cycle model In this model, the pheromone trails are updated globally using the following formula:

$$\tau_{ij}(t + n) = \rho\tau_{ij}(t) + \Delta\tau_{ij}(t, t + n) \tag{10.10}$$

where

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/T_k & \text{if the } k\text{th ant uses edge } (i, j) \text{ in its tour} \\ & \text{(between time } t \text{ and } t + n) \\ 0 & \text{otherwise} \end{cases} \tag{10.11}$$

where T_k is the length of the tour taken by the kth ant.

In the ant-density and ant-quantity models, ants locally update the pheromone trail directly after moving from one state to the adjacent one, whereas in the ant-cycle model, the pheromone trail is updated only after all the ants have constructed their tours and the amount of pheromone deposited by each ant is set to be a function of the tour quality (Dorigo et al. 1996).

Ant system with elistic ants In this ant system for the ant-cycle model, an elistic ant is one that reinforces the edges belonging to T^+—the best tour found from the beginning of the trial—by a quantity Q/L^+, where L^+ is the length of T^+. During each cycle, e elistic ants are added to the usual ants, so that the edges belonging to T^+ get extra reinforcement eQ/L^+. The idea is that the pheromone trail of T^+ so reinforced will direct the search of the other ants in all probability towards a solution composed of some edges of the best tour itself. So, the pheromone trail is globally updated as follows (Bonabeau et al. 1999):

$$\tau_{ij}(t) \leftarrow \rho\tau_{ij}(t) + \Delta\tau_{ij}(t) + e\Delta\tau^{e}_{ij}(t) \tag{10.12}$$

where

$$\Delta\tau^{e}_{ij}(t) = Q/L^{+} \tag{10.13}$$

Max-min ant system The modification introduced by the max-min AS with respect to other ASs are the following.

(1) To exploit the best solutions found after all the ants have constructed a solution, only one ant is allowed to add pheromone:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau^{\text{best}}_{ij} \tag{10.14}$$

where $\Delta\tau^{\text{best}}_{ij} = 1/T^{\text{best}}$, the ant that is allowed to add pheromone is the globally best solution of T^+.

(2) To avoid search stagnation, the allowed range of the pheromone trail strength is limited to the interval $[\tau_{\min}, \tau_{\max}]$, that is, $\forall\tau_{ij}, \tau_{\min} \le \tau_{ij} \le \tau_{\max}$.

(3) The pheromone trails are initialized to the upper ($\tau_{\max}$) trail, which causes higher exploration at the start of the algorithm (Dorigo & Stützle et al. 1999).

AS ranking model In the AS ranking model, the globally best tour is always used to update pheromone trails, similar to the elistic strategy of ASs. Additionally, a number of the best ants of the current iteration are allowed to add pheromone. With this aim, ants are sorted by tour length $[L^1(t) \le L^2(t) \le \ldots \le L^m(t)]$, where m is the total number of ants in the current iteration, and the quantity of pheromone an ant may deposit is weighted according to the rank r of the ant. Only the $w-1$ best ants of each iteration are allowed to deposit pheromone. The globally best solution, which gives the strongest feedback, is given the weight w. The rth best ant of the current iteration contributes to pheromone updating with a weight given by $\max(0, w-r)$. Thus, the modified update rule is (Dorigo & Stützle et al. 1999)

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \sum_{r=1}^{w-1}(w-r)\Delta\tau^{r}_{ij}(t) + \Delta\tau^{gb}_{ij}(t) \tag{10.15}$$

where

$$\Delta\tau^{r}_{ij}(t) = 1/L^{r}(t)$$

$$\Delta T^{gb}_{ij}(t) = 1/L^{gb} \tag{10.16}$$

Ant colony system In an ACS, the global trail-updating rule is applied only to the edges belonging to the best tour since the beginning of the trail. The updating rule used is

$$\tau_{ij}(t) \leftarrow (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t) \tag{10.17}$$

where the (i, j)'s are the edges belonging to T^+—the best minimum length tour since the beginning of the trial, ρ is a parameter governing pheromone decay, and the change in pheromone concentration is given by

$$\Delta\tau_{ij}(t) = 1/L^+ \tag{10.18}$$

where L^+ is the cost of T^+. This procedure allows only the best tour to be reinforced by a global update. Local updates are also performed, so that other solutions can emerge. A local update is performed as follows: when, while performing a tour, ant k is in state i and selects a state $j \in J_i^k$ [Eqns (10.4) and (10.5)], the pheromone concentration of (i, j) is updated using the following formula:

$$\tau_{ij}(t) \leftarrow (1-\rho)\tau_{ij}(t) + \rho\tau_0$$

where τ_0 is the initial value of the pheromone trails.

The differences between an ACS and an AS are the following.

(1) An ACS uses a more aggressive action rule than an AS.
(2) Pheromone in an ACS is added only to arcs belonging to the globally best solution.
(3) Each time an ant uses an arc (i, j) to move from city i to city j in ACS, it removes some pheromone from the arc.

In an ACS, both local trail updating and global trail updating are used. In local trail updating, m artificial ants are placed on randomly selected cities and moved to new cities in each time step; the pheromone trail on the edges used are modified. In global trail updating (which is done by adding an amount of pheromone that is inversely proportional to the tour length), after all the ants have completed a tour, the ant that made the shortest tour modifies the edges belonging to its tour (Bonabeau et al. 1999).

Recruiting ant colony system In this model, there exist two kinds of agents with respect to the real biological pattern (Stamer 2001).

- *Search ants*: The primary task of these tasks is to search the surroundings randomly for food. For this purpose, the common action choice rule of ACSs mentioned above is ideal.
- *Transport ants*: These ants wait in their nests until an acceptable food source is found, and then start transporting the food. To do this, they follow the route of the recruiting search ants with very high probability. When transport ants happen to lose track, they fall back upon pheromones for reorientation. Sometimes they change their 'recruit' label too. Of course, the number of transport ants must be set much larger than the number of

search ants, particularly to let the recruiting strategy work. This model has got a few similarities to the AS ranking model and the AS with elistic strategy; however, some new aspects like group recruiting, changing the recruit label, etc. are incorporated in it, which result in better performance.

10.7.4.2 Engineering applications

Ant colony intelligence has been adopted in many engineering applications in robotics, VLSI design, telecommunication network routing, operational research, image processing, data mining, power system applications, business management, etc.—wherever an optimal decision has to be achieved. In this section we study two problem-solving examples using ant intelligence—the travelling salesman problem (Simon et al. 2003) and the hard combinatorial unit commitment problem (UCP), which is a power system application.

The travelling salesman problem In this application, our objective is to solve some of the standard test problems available in the TSPLIB Benchmark library (http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/tsp/) and compare the solutions with few of the ant system models presently available. The background of this problem has already been discussed in Section 10.7.4. However, here we will consider only the symmetric TSP problem, which is a static combinatorial optimization problem with N cities or nodes.

Let us consider m total artificial ants for solving the TSP. Each artificial ant is like an agent that follows certain rules. Initially, each of the m ants is placed in a randomly or uniformly chosen city, where it applies a state transition rule to choose a city it has to go to, taking care of the constraints of the problem. An ant constructs a tour as follows. At a city i, it chooses a still unvisited city j probabilistically, biased by the pheromone trail strength $\tau_{ij}(t)$ on the arc between city i and city j and locally available heuristic information, which is a function of the arc length. Ants probabilistically prefer cities which are close and are connected by arcs with high pheromone trail strength.

To construct a feasible solution, each ant is assigned a limited amount of memory, called a *tabu list*, in which the current partial tour is stored (Dorigo et al. 1996). This memory is used to determine at each construction step the set of cities still to be visited and to guarantee a feasible solution. Additionally, it allows the ant to retrace its tour, once completed. After all ants have completed their tours, the pheromones are updated. This is typically done by first reducing the pheromone trail strengths by a constant factor and then allowing the ants to deposit pheromone on the arcs they have visited. The trails are updated in such a way that the arcs contained in shorter tours and/or visited by many ants receive a higher amount of pheromone and are, therefore, chosen with higher probability in the following iterations of the algorithm. In this sense, the amount of pheromone $\tau_{ij}(t)$ represents the learned desirability of choosing the next city j when an ant is in city i (Dorigo & Gambardella 1997; Dorigo & Stützle 1999). The TSP problem can be implemented the following algorithmic steps.

(1) At time $t = 0$, distribute m ants uniformly or randomly, position them in different cities, and set their initial value $\tau_{ij}(0)$ to a small positive constant c. Set the first element of each ant's tabu list to its starting city.

(2) At time $t + 1$, move each ant from city i to j according to the probabilistic transition rule given in Eqn (10.3) in accordance with the tabu list maintained for that particular tour. Continue this for n iterations to complete an entire tour or cycle.

(3) Save the minimum distance covered among all the tour distances covered by ants for that particular tour or when one cycle is completed.

(4) After each complete tour, update the pheromone trail as follows.

(a) Let τ_{ij} be the trail intensity on edge (i, j) at time t and let each ant simultaneously choose a city at time t and move to the next city at time $t + 1$. An iteration is the completion of an ant's move from i to j. For every n iterations of the algorithm, an ant completes one tour, called a cycle.

(b) Update the pheromone trail globally with the help of Eqns (10.6), (10.8), (10.10), and (10.12) for the ant-quantity, ant-density, ant-cycle, and ant system with elistic ants models, respectively. ρ is the evaporation factor between discrete time steps, $0 \leq \rho < 1$. Here

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

where $\Delta\tau_{ij}^{k}$ is the change in pheromone concentration, which is different for all the four AS models represented by Eqns (10.7), (10.9), (10.11), and (10.13).

(5) Repeat steps (2) to (4) until there is no improvement in the shortest tour saved.

(6) Display the shortest tour.

It is found that after a required number of cycles, there is more pheromone deposition in the paths that are visited frequently by the ants; these paths will constitute the shortest tour. Now all the ants make the same tour, which is stagnating behaviour because it denotes a situation in which the algorithm stops searching for alternative solutions.

Parameter settings Good convergence behaviour for all the ACMs can only be obtained by selecting optimal control parameters (Dorigo & Stützle 2001). Parameters such as $\alpha, \beta, \rho, Q, e$, and the initial trail level τ_0 affect directly or indirectly the computation of the probability transition rule of Eqn (10.3). The program is run three times and the results are averaged for each version of the AS model to get some statistical information. Here the Eilon 30-city problem is considered. With a default setting of the parameters taken initially, one of the parameters is varied and the others kept constant. In the initialization phase, the ants are randomly distributed to various positions. It is also compared with the

uniform distribution of ants for the ant system with elistic ants model. In all these models, for a particular setting of the parameters, it is observed that there is a monotonic increase/decrease of the cost, and after this, the average length starts to decrease/increase. This can be understood from the parameter settings given in Table 10.2 (AL denotes average length, MSD denotes mean standard deviation), where the control parameters, which are selected for the individual AS models, are shown in bold. It has also been observed during initialization that the random and uniform distributions of ants are performed almost similarly; however, random initialization consistently yields the best tour lengths as compared to uniform distribution, which can be understood with the help of Table 10.3.

Table 10.2 Parameter settings for AS models (default settings: $\alpha = 1$, $\beta = 1$, $\rho = 0.7$, $m = n$, $Q = 100$, $e = 5$)

(a) Ant-quantity model

α	0	0.5	**1**	2	5
AL	386.65	320.39	**310.54**	329.85	346.14
MSD	47.57	32.87	**29.51**	2.57	29.14

β	0	1	2	**5**	10
AL	412.06	317.37	309.82	**305.36**	318.51
MSD	3.81	22.84	24.39	**17.78**	13.72

ρ	0.3	0.5	**0.7**	0.9
AL	317.37	312.62	**319.28**	331.38
MSD	19.84	17.92	**21.41**	4.56

Q	1	**100**	10,000
AL	327.62	**310.54**	323.41
MSD	13.18	**21.17**	16.58

m	10	**30**	50
AL	334.17	**310.54**	323.58
MSD	18.60	**21.17**	18

(b) Ant-density model

α	0	0.5	**1**	2	5
AL	401.90	328.02	**319.99**	366.01	346.20
MSD	47.69	36.94	**25.89**	0.77	0.77

(contd)

Table 10.2 ***(contd)***

β	0	1	2	5	10
AL	649.78	339.31	316.12	**305.36**	314.36
MSD	29.50	23.69	22.32	**17.52**	16.25

ρ	0.3	0.5	0.7	0.9
AL	336.88	356.78	**337.36**	358.96
MSD	29.35	26.07	**25.59**	21.71

Q	1	100	10,000
AL	**318.71**	353.34	336.29
MSD	**7.40**	20.48	31.03

m	10	30	50
AL	351.07	**337.25**	348.59
MSD	16.88	**26.34**	2.39

(c) Ant-cycle model

α	0	0.5	1	2	5
AL	387.99	335.21	**325.71**	348.50	360.19
MSD	47.69	39.32	**28.16**	0.91	13.06

β	0	1	2	5	10
AL	619.11	333.89	306.74	**305.36**	305.36
MSD	25.29	24.79	22.09	**18.89**	23.37

ρ	0.3	0.5	0.7	0.9
AL	323.60	341.16	**320.35**	336.58
MSD	31.40	23.56	**20.12**	26.27

Q	1	100	10,000
AL	331.23	**336.58**	323.90
MSD	28.97	**26.27**	24.79

m	10	30	50
AL	362.74	**336.58**	340.31
MSD	15.29	**26.27**	23.15

(contd)

Table 10.2 *(contd)*

(d) Ant-cycle model with elistic ants

α	0	**0.5**	1	2	5
AL	382.15	**314.91**	317.08	356.97	400.64
MAD	48.12	**34.42**	29.19	5.451	2.34

β	0	1	2	**5**	10
AL	557.51	317.08	306.74	**305.36**	310.05
MAD	35.95	27.28	9.50	**4.22**	3.58

ρ	0.3	0.5	**0.7**	0.9
AL	322.62	328.34	**317.08**	326.86
MAD	25.53	27.71	**29.19**	14.27

Q	1	**100**	10,000
AL	320.59	**317.08**	327.26
MAD	25.50	**28.19**	24.73

m	10	**30**	50
AL	378.21	**317.08**	318.83
MAD	14.50	**29.19**	8.92

e	1	3	**5**	7	10	20
AL	317.27	317.85	**317.08**	347.59	334.77	351.39
MAD	32.00	27.61	**29.19**	27.83	12.81	15.07

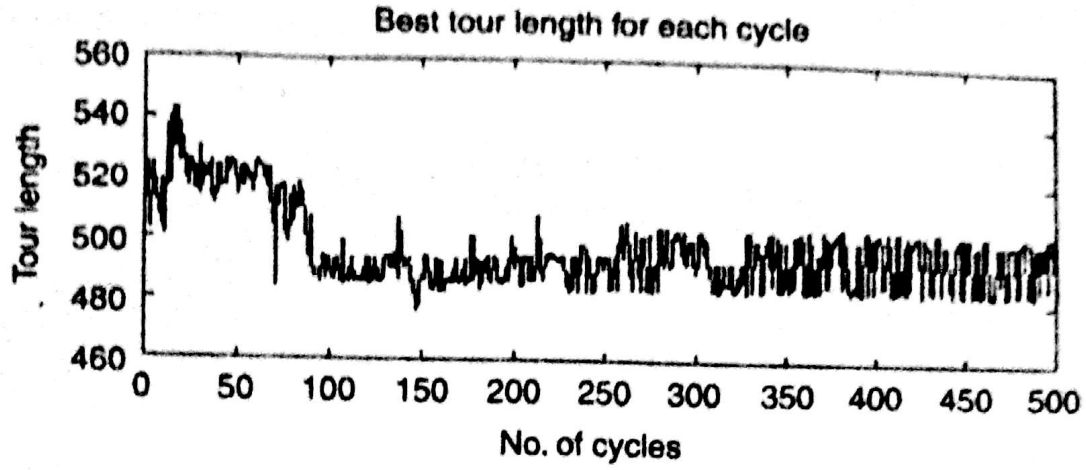
Table 10.3 Random distribution (RD) vs uniform distribution of ants (UD) (*N*th cycle: the cycle at which minimum convergence is obtained)

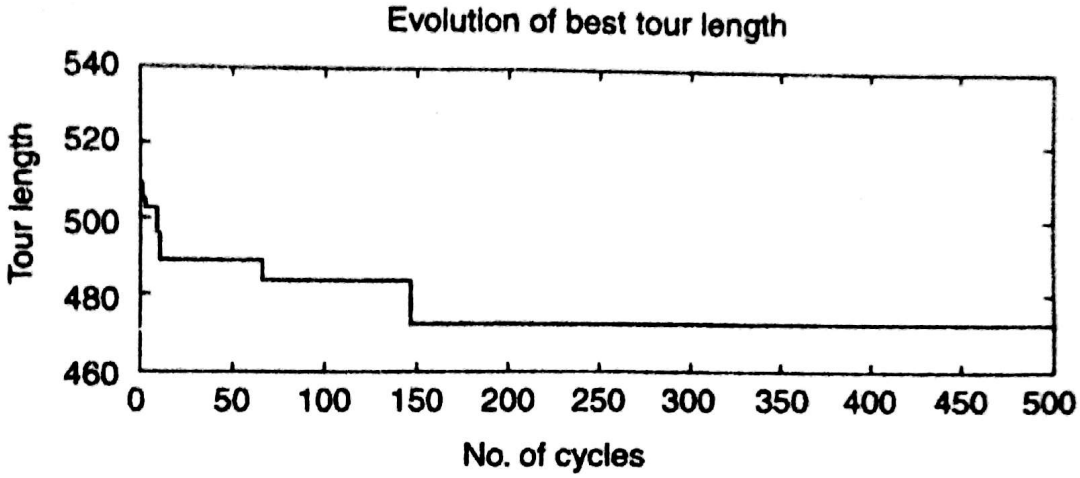
RD			*UD*		
AL	*MSD*	*Nth cycle*	*AL*	*MSD*	*Nth cycle*
305.36	4.22	4	305.36	3.18	2
305.36	2.42	4	309.69	19.51	146
305.36	4.26	3	305.36	4.70	4
305.36	1.03	1	309.69	19.51	146

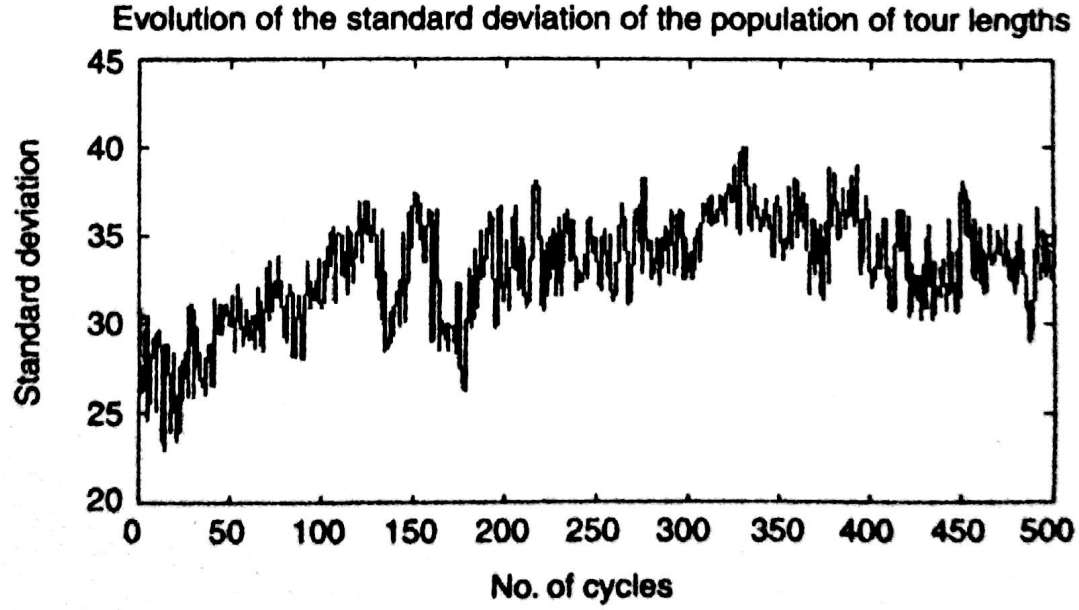
Comparison of AS models For this comparison, the control parameters are selected for each model and tested for Burma 14 cities, Eilon 30 cities, Eilon 51 cities, and a 70-city problem (Smith/Thompson) present in the TSPLIB; the results are shown in Table 10.4. The resultant plots for the Eilon 51-city problem are shown in Figs 10.4–10.7, where ST denotes the shortest tour and TL denotes the tour length.

Table 10.4 Comparison of ant system models (*N*th cycle: the cycle at which minimum convergence is obtained).

Problem dimension	*TL*	*MSD*	*Total time (s)*	*Nth cycle*
Ant-quantity				
Bur 14	32.06	1.89	98	2
Eilon 30	305.36	17.81	924	4
Eilon 51	476.47	27.32	5190	145
ST 70	753.35	39.95	14,600	11
Ant-density				
Bur 14	31.46	2.08	99	4
Eilon 30	305.36	15.46	916	176
Eilon 51	476.12	33.21	5220	156
ST 70	741.82	46.69	14,300	20
Ant-cycle				
Bur 14	31.46	1.67	97	52
Eilon 30	305.36	18.89	125	4
Eilon 51	474.44	26.2	5200	3
ST 70	730.97	39.86	15,400	22
Ant system with elistic ants				
Bur 14	31.23	1.92	95	5
Eilon 30	305.36	4.22	105	4
Eilon 51	455.93	31.06	5086	47
ST 70	706.76	50.18	14,500	215



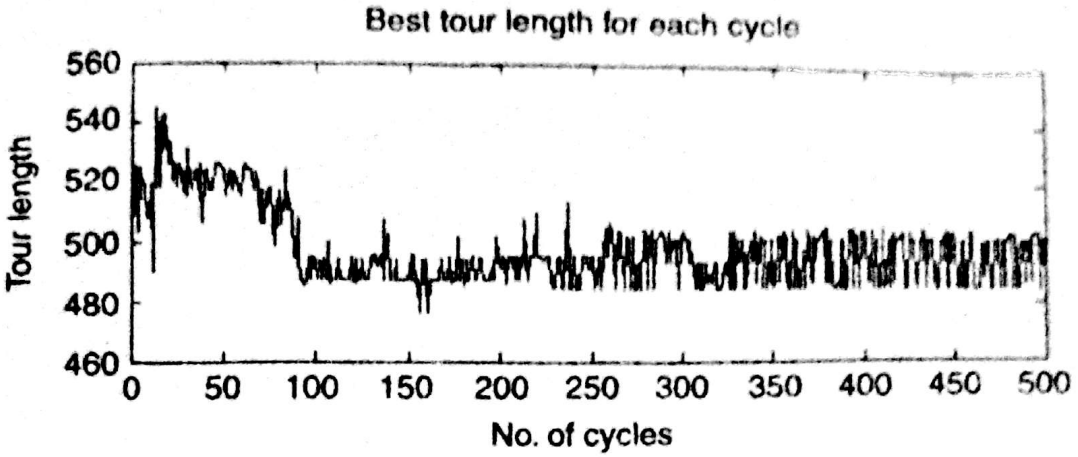


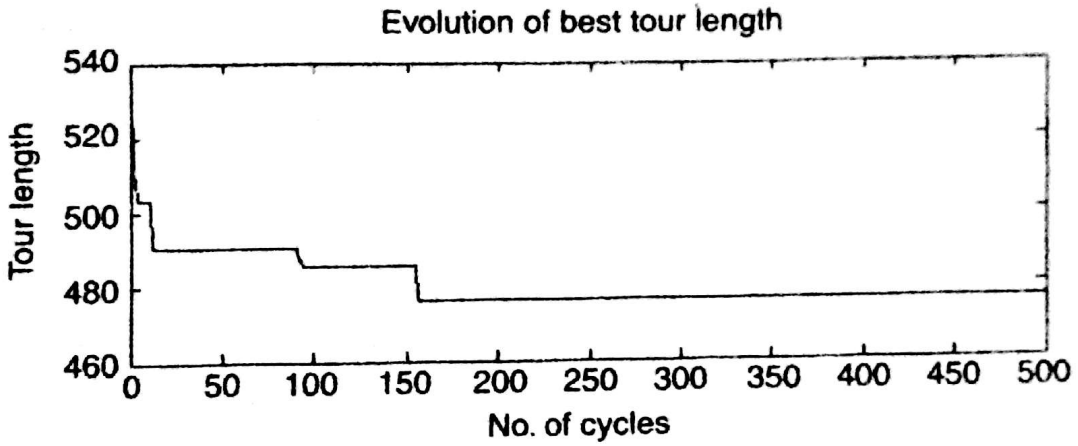


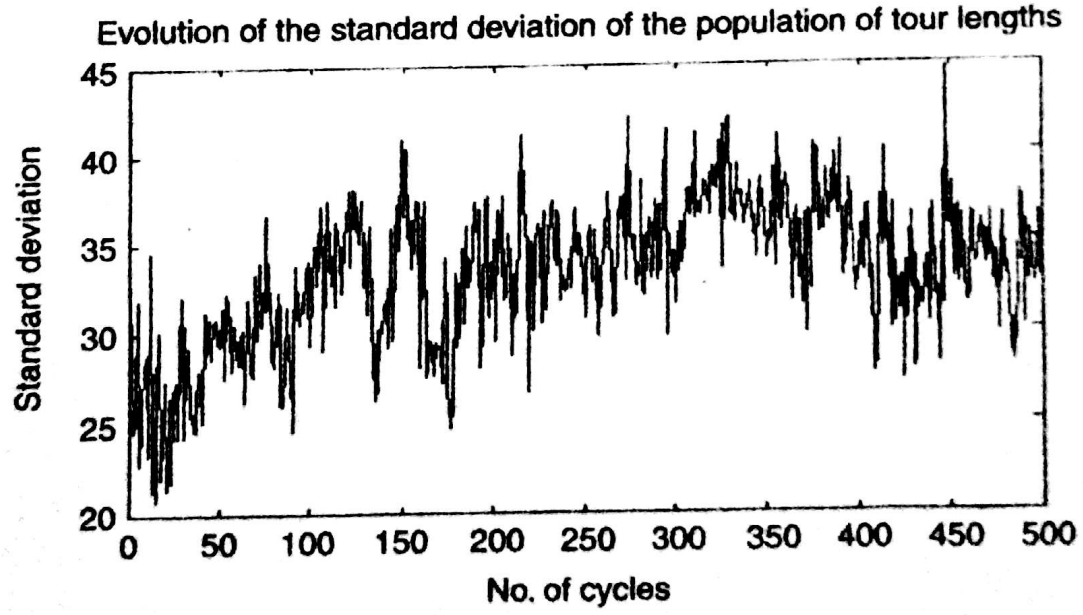
ST= {40, 13, 41, 19, 42, 44, 15, 37, 17, 4, 18, 47, 12, 46, 51, 27, 1, 22, 2, 16, 50, 9, 49, 5, 38, 11, 32, 48, 6, 14, 25, 43, 24, 23, 7, 26, 8, 31, 28, 3, 20, 35, 36, 29, 21, 34, 30, 10, 39, 33, 45, 40}

TL = 476.47

Fig. 10.4 Ant-quantity model



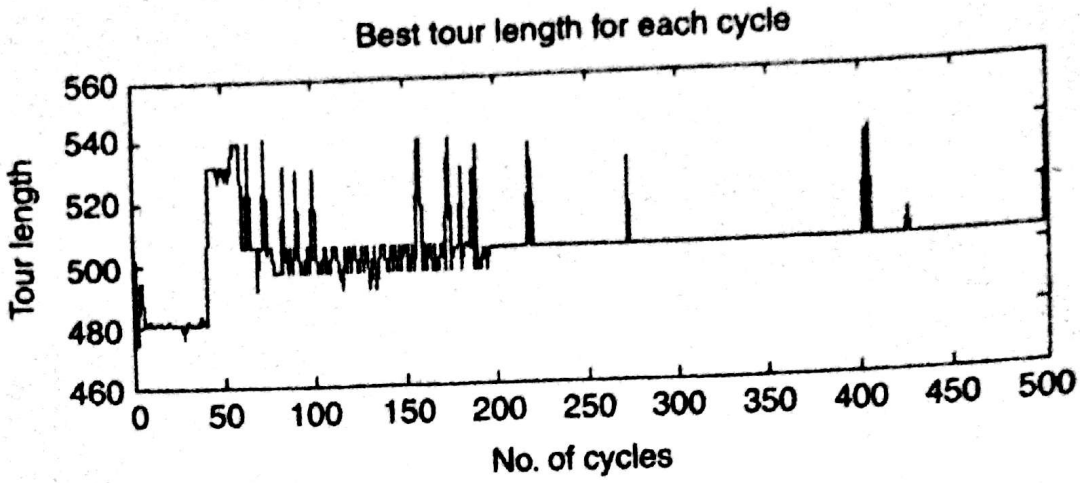


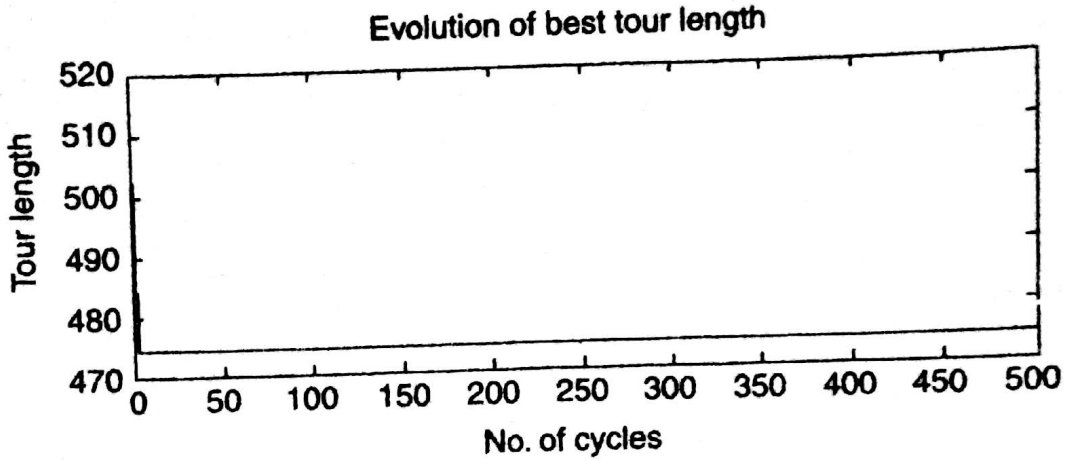


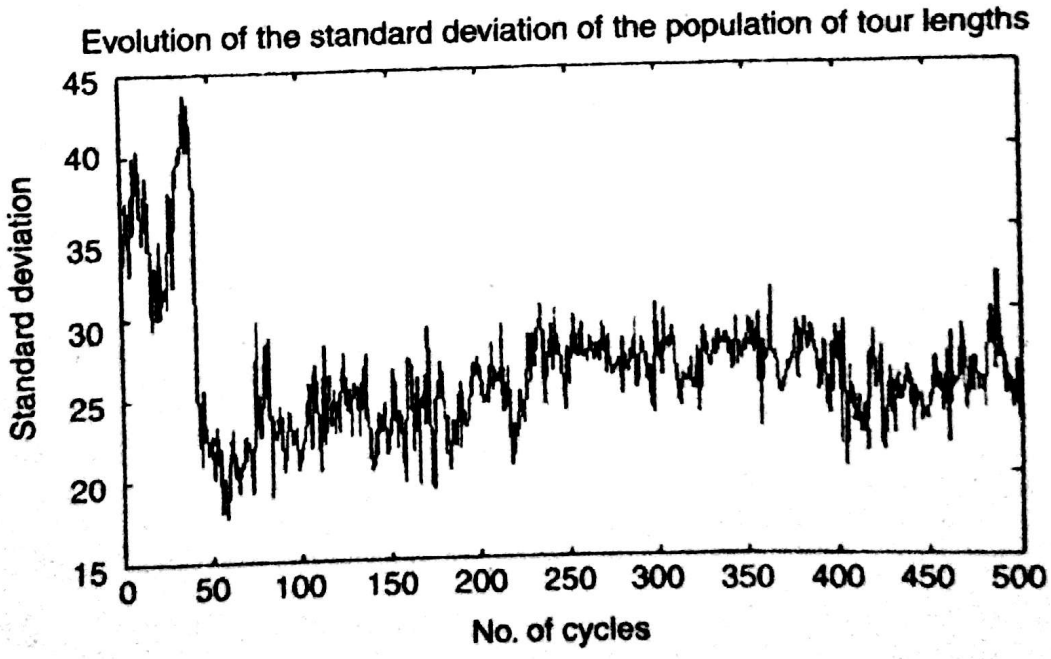
ST= {25, 14, 6, 27, 1, 22, 2, 16, 50, 9, 49, 5, 38, 11, 32, 51, 46, 12, 47, 18, 4, 17, 37, 15, 44, 42, 19, 41, 13, 40, 45, 33, 39, 10, 30, 34, 21, 29, 20, 35, 36, 3, 28, 31, 8, 26, 7, 23, 24, 43, 48, 25}

TL = 476.12

Fig. 10.5 Ant-density model



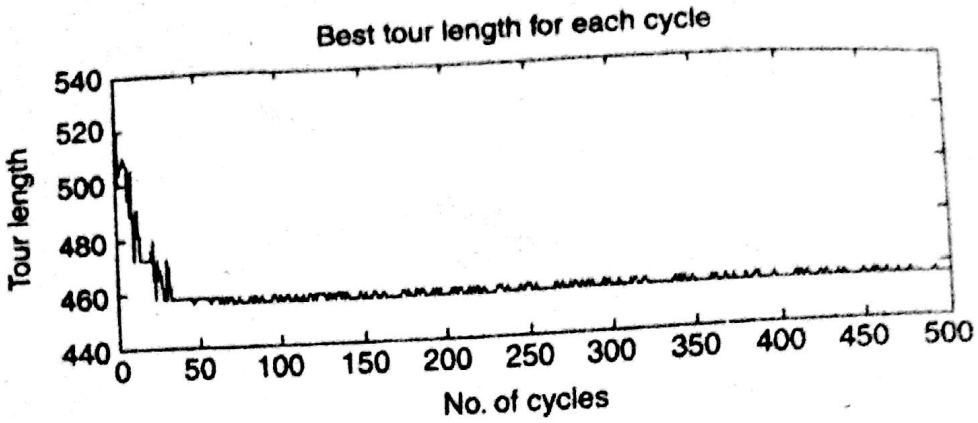


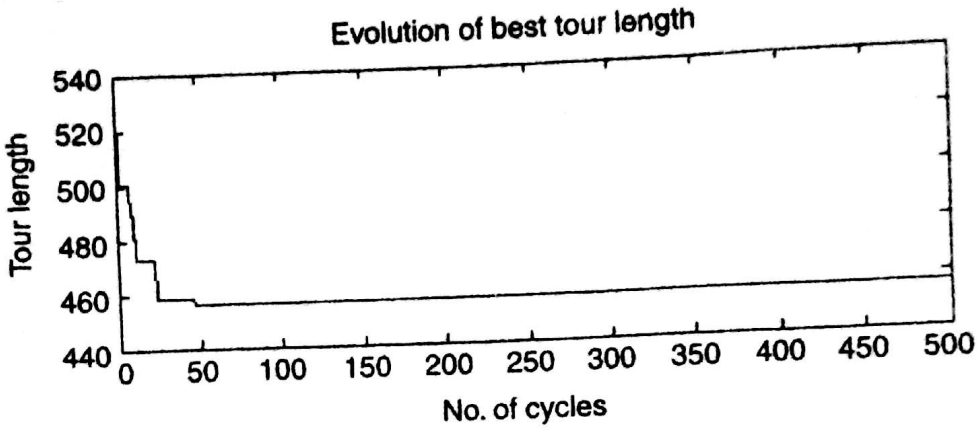


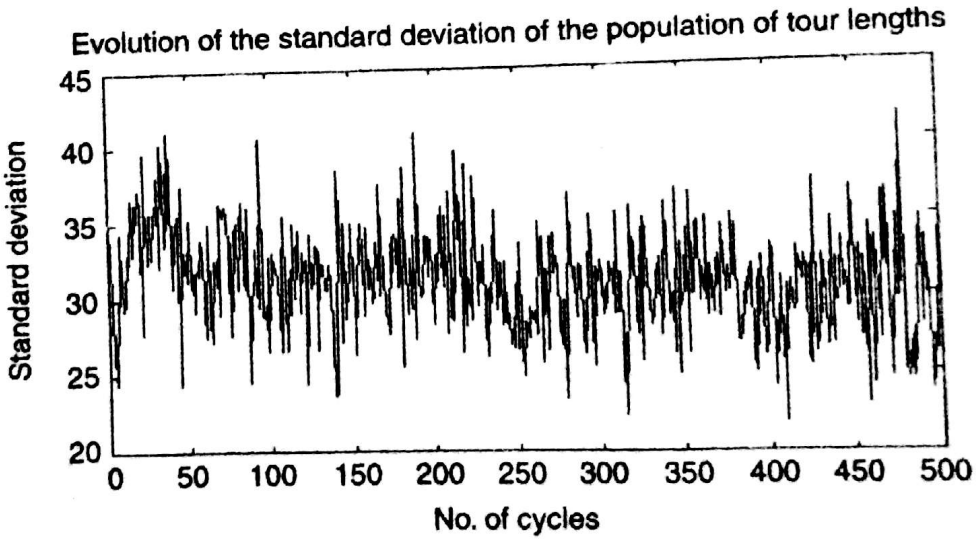
ST= {23, 7, 26, 8, 31, 28, 3, 20, 35, 36, 22, 1, 32, 11, 38, 5, 49, 9, 50, 16, 2, 29, 21, 34, 30, 10, 39, 33, 45, 15, 44, 37, 17, 4, 18, 47, 12, 46, 51, 27, 6, 14, 25, 13, 41, 19, 42, 40, 24, 43, 48, 23}

TL = 474.44

Fig. 10.6 Ant-cycle model







ST= {24, 23, 7, 26, 8, 31, 28, 3, 36, 35, 20, 21, 34, 30, 9, 50, 16, 2, 22, 1, 32, 11, 38, 5, 49, 10, 39, 33, 45, 44, 15, 37, 17, 4, 18, 14, 25, 13, 40, 42, 19, 41, 47, 12, 46, 51, 27, 6, 48, 43, 24}

TL = 455.93

Fig. 10.7 Ant system with elistic ants model

The results give a clear picture of how suboptimal paths are improved when the control parameters of good convergent behaviour are used for these models. Also, the mean standard deviation of the population of the tour lengths of the ants for each cycle is varying even after 500 cycles, which reveals that the ACSs are still trying to search for better solutions and indicates the potential of these AS models.

The results obtained prove that the ant system with elistic ants model is the best among the four models studied and that the total time taken increases as the dimension of the problem increases. For problems of small dimension, all the four models faired good, but this is not so when the dimension increases. The hardware platform on which these models have been implemented is Pentium IV, 2 GHz, 256 MB RAM using the MATLAB 6.1 simulation environment.

The unit commitment problem The need to obtain a globally optimum solution in the highly non-linear and computationally difficult power system environment has been a focus of research. The UCP is a hard combinatorial optimization problem aimed at determining the optimum schedule of generating units (i.e., switching on and off of N generating units over a period of time for the demand forecasted to be served) by minimizing the overall cost of power generation while satisfying a set of system constraints. Finding a good solution to the UCP in reasonable amount of time is very critical since it could mean significant annual financial savings in power generating costs. To 'commit' a generating unit is to 'turn it on', that is, to bring the unit up to speed, synchronize it to the system, and connect it so that it can deliver power to the network. The problem of 'committing enough units and leaving them online' is one of the economics. It is quite expensive to run too many generating units. A great deal of money can be saved by turning units off (de-committing them) when they are not needed. The generic UCP can be formulated as to minimize operational cost subject to minimum up time and down time constraints, crew constraints, ramp constraints, unit capability limits, deration of units, unit status, generation constraints, and reserve constraints (Wood & Wollenberg 1984; Sheble & Fahd 1994; Padhy 2004).

For the last two and a half decades, research work is being carried out in the area of the unit commitment problem. The major limitations of the UCP lie in the problem's dimensions, large computational time, and complexity in programming and optimum results. Since optimum schedules for committing units can save millions of dollars per year in production costs, efforts are being made to solve the UCP using simulated annealing, expert systems, artificial neural networks, fuzzy systems, genetic algorithm, evolutionary programming, and hybrid models (Padhy 1994). In this section, we solve the UCP with the help of the ant colony system model. The generic ACM-based unit commitment problem can be formulated as follows:

Minimize the operational cost (OC)

$$OC = \sum_{i=1}^{N} \sum_{t=1}^{T} FC_{it}(P_{it}) = ST_{it} + SD_{it} \text{ (in \$/hr)} \tag{10.19}$$

where $FC_{it}(P_{it})$ (fuel cost) is the input/output (I/O) curve that is modelled with a polynomial curve (normally a quadratic function):

$$FC_{it}(P_{it}) = a_i P_{it}^2 + b_i P_{it} + c_i \text{ (in \$/hr)} \tag{10.20}$$

where a_i, b_i, and c_i are cost coefficients and P_{it} is the power generated of unit i during time period t in MW. The start-up cost is described by

$$\mathrm{ST}_{it} = \mathrm{TS}_{it}F_{it} + (1 + e^{(D_{it}\,\mathrm{AS}_{it})})\mathrm{BS}_{it}F_{it} + \mathrm{MS}_{it} \text{ (in \$/hr)} \tag{10.21}$$

where TS_{it} is the turbine start-up cost, BS_{it} is the boiler start-up cost, MS_{it} is the start-up maintenance cost, D_{it} is the down time in hours, and AS_{it} is the boiler cool-down coefficient. Similarly, the shutdown cost is described by

$$\mathrm{SD}_{it} = kP_{it} \text{ (in \$/hr)} \tag{10.22}$$

where k is the proportional constant.

Subject to the following constraints:

Minimum up time $0 < T_{iu} \leq$ number of hour units for which G_i has been online where T_{iu} is the minimum up time.

Minimum down time $0 < T_{id} \leq$ number of hour units for which G_i has been off-line

where T_{id} is the minimum down time. The maximum and minimum output limits on the generators are as follows:

$$P_{it(\min)} \leq P_{it} \leq P_{it(\max)}$$

Those on the power rate are

$$\nabla P_{it(\min)} \leq \nabla P_{it} \leq \nabla P_{it(\max)}$$

where ∇P_{it} is the power rate of generator i (in MW/hr). The power balance is represented as

$$\sum_{i=1}^{N} P_{it} = \mathrm{load}(H)$$

where load(H) is the system load at hour H.

Let us take the numerical example of the four-unit system given in Table 10.5.

The unit characteristics for the four-unit system are as follows. Initial unit status: Hours off-line (–) or online (+). The fuel cost equations are

$$C_1 = 25 + 1.5000P_1 + 0.00396P_1^2$$

$$C_2 = 72 + 1.3500P_2 + 0.00261P_2^2$$

$$C_3 = 49 + 1.2643P_3 + 0.00289P_3^2$$

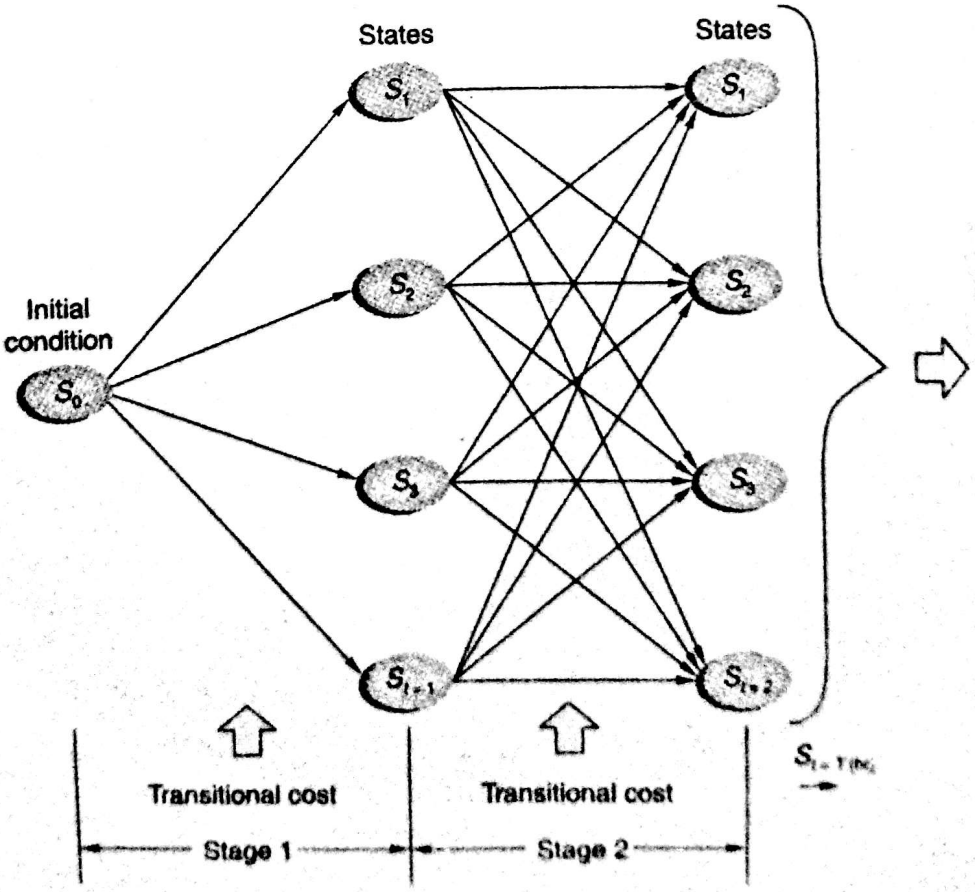
$$C_4 = 15 + 1.4000P_4 + 0.00510P_4^2$$

The ACS approach for solving the UCP consists mainly of two phases. In the first phase, all the possible S_t states of the tth hour that satisfy the load demand are determined; this is continued for the complete scheduling period of 24 hours, which constitutes the ant search space (ASS). An ASS involving multi-decision states is seen in Fig. 10.8. Here, in the initialization part, the forecasted load demands and other relevant problem data from the system are taken for computation. Economic

dispatch (ED) using the Lagrange multiplier method is used, which calculates the generator output and the production cost for each hour. The exhaustive enumeration technique is used to find all the possible combinations of the generating units available.

Table 10.5 Numerical example of the four-unit system

	Unit 1	*Unit 2*	*Unit 3*	*Unit 4*
$P_{it(\max)}$ (MW)	80	250	300	60
$P_{it(\min)}$ (MW)	25	60	75	20
Ramp level (MW/hr)	16	50	60	12
Minimum up time (hr)	4	5	5	1
Maximum down time (hr)	2	3	4	1
Shutdown costs ($)	80	110	300	0
Start-up costs ($)				
Hot	150	170	500	0
Cold	350	400	1100	0.02
Cold start time (hr)	4	5	5	0
Initial unit status	–5	8	8	–6



* $S_{t(w)}$ is the eligible states satifying load demand for the tth hour.

Fig. 10.8 A multi-decision search space

Once the search space is identified, the second phase involves the artificial ants that are allowed to pass continuously through the ASS. Each ant starts its journey from the initial condition, termed the starting node, reaches the end stage, and finally vanishes. So, it is a continuous flow of ants, and they never return. Once an ant reaches the end stage, a tour is completed and the overall generation cost path is generated. For each stage of the time period t, the ant selects an operational cost (OC) calculated for all N generator units in order to minimize the overall OC. This is continued till the time period becomes T and a tour is completed by that particular ant. Whenever a tour is completed by an individual ant and the total generation cost is found to be less than the minimum cost paths taken by the previous ants, the present cost path is captured. This procedure is continued for all the remaining ants available at the starting node, which enables one to trace the optimal path.

The ant colony search mechanism can be divided into four parts: (a) initialization, (b) transition rule, (c) pheromone trail update rule, and (d) parameter settings (see Fig. 10.9).

Initialization During initialization, the parameters—requisite number of ants, relative importance of the pheromone trail, relative importance of the visibility, initial available pheromone trail, a constant related to the quantity of the trail laid by the ants, evaporation factor, tuning factor, number of elistic ants, etc.—have to be fixed and taken care of, which will be explained in detail in parameter settings.

Probability transition rule The probability transition rule for the ACS is the same as Eqns (10.4) and (10.5) except that here the heuristic function $\eta_{ij(\text{or})iu} = 1/C_{ij}$, where C_{ij} is the transitional cost for that particular stage.

Pheromone trail update rule Once the ants start choosing the minimum cost states, the pheromone trail update rule has to be implemented. In ACSs, the global trail-updating rule is the same as Eqn (10.7), and here $\Delta\tau(t) = 1/C^+$, where C^+ is the cost of T^+, the best minimum cost tour.

Parameter settings In the ACS model, the parameters β, ρ, and q_0 are those that affect the probability transition rule. The numerical example of the four-unit system for the 24-hr scheduling period is tested for each parameter taking several values, all the others being constant (default settings; in each experiment only one of the values is changed), over ten simulations in order to achieve some statistical information about the average evolution. The initial trail level τ_0 is always set to 1. In the case of the ACS model, $\alpha = 1$. The number of ants allowed to pass through the ASS is 50. Table 10.6 shows the average total generation costs (TGCs) for the various parameter settings.

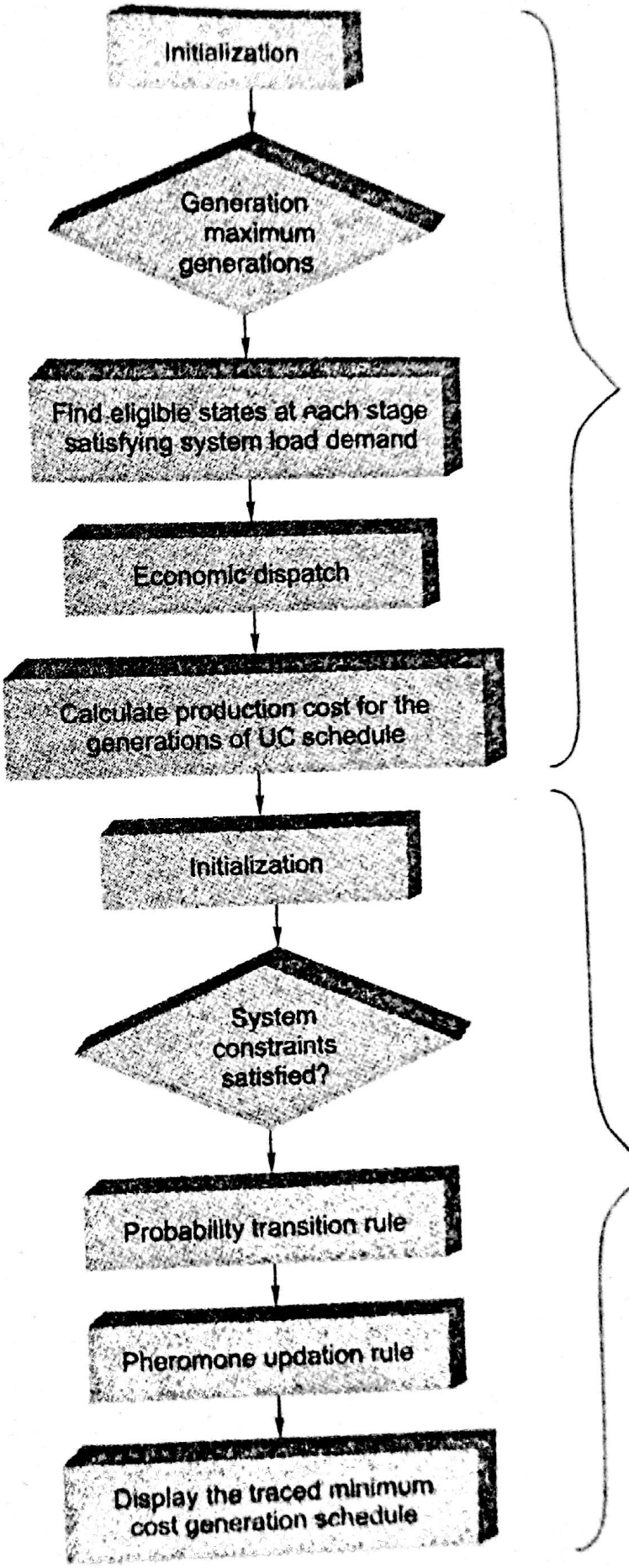


Fig. 10.9 ACS-based UCP flow chart

Table 10.6 Parameter settings for the ACS model (default settings: $\beta = 1$, $\rho = 0.3$, $\tau_0 = 1$, $q_0 = 0.7$)

β	*Av. TGC ($/day)*	ρ	*Av. TGC ($/day)*	q_0	*Av. TGC ($/day)*
0	27,324.25	0	27,568.67	0.1	27,937
1	27,224.5	**0.1**	**27,149.25**	0.3	27,685
2	27,206	0.2	27,170.25	0.5	2.7478
5	27,115	0.3	27,328	0.7	27,328
7	27,107.5	0.5	27,373.5	**0.9**	**27,135**
10	**27,030.25**	0.7	27,442.67		
15	27,151.5				

The results of the experiment show that in this model, a high value of α means that the trail is very important, and therefore ants tend to choose edges chosen by other ants in the past. This is true until the value of β becomes very high: in this case even if there is high amount of pheromone on a trail, there is always a high probability of an ant choosing another trail of lower cost. The ACS model performs well when the tuning factor is set to a high value, indicating that the model incorporates higher exploitation of the knowledge available about the problem. High values of β and/or low values of α make the algorithm very similar to a stochastic *multigreedy* algorithm.

The evolution of the convergence of the minimum cost path can be seen from Fig. 10.10—after the minimum number of ants have passed through the ant search space and completed their tours, the convergence of the cost finally becomes stagnant. Even after the stagnant situation, ants still try to explore any of the minimum cost paths that are available, which indicates the strength of the ant colony system model. This can be understood from Fig. 10.11 by the oscillations for each individual ant taking its tour.

The test results indicate that an ant can absorb information from the experience gained from its fellow ants' behaviour to get the best results, which are almost globally optimum. Even though at one transition stage, the cost of the ACS approach may be high as compared to dynamic programming (DP) approach, there is an overall reduction of generation cost because the ants can still foresee and pick out the stages coming afterwards which can belong to a low-cost path. It can be also seen from the literature that the DP approach is even now considered the best choice; however, the above results show that the ACS approach is performing successfully and helps to reduce the TGC more efficiently. The TGC for the four-unit system obtained using the DP approach is 27,541 $/day. Therefore, a saving of 2.2367% is obtained using the ACS model. A comparison of the transitional cost for the conventional dynamic programming and the ACS approach can be observed in Fig. 10.12 and Table 10.7. The hardware platform that has been used to implement this model is AMD Athlon[tm] XP 1800 + 1.54 GHz, 224 MB of RAM using the MATLAB environment.

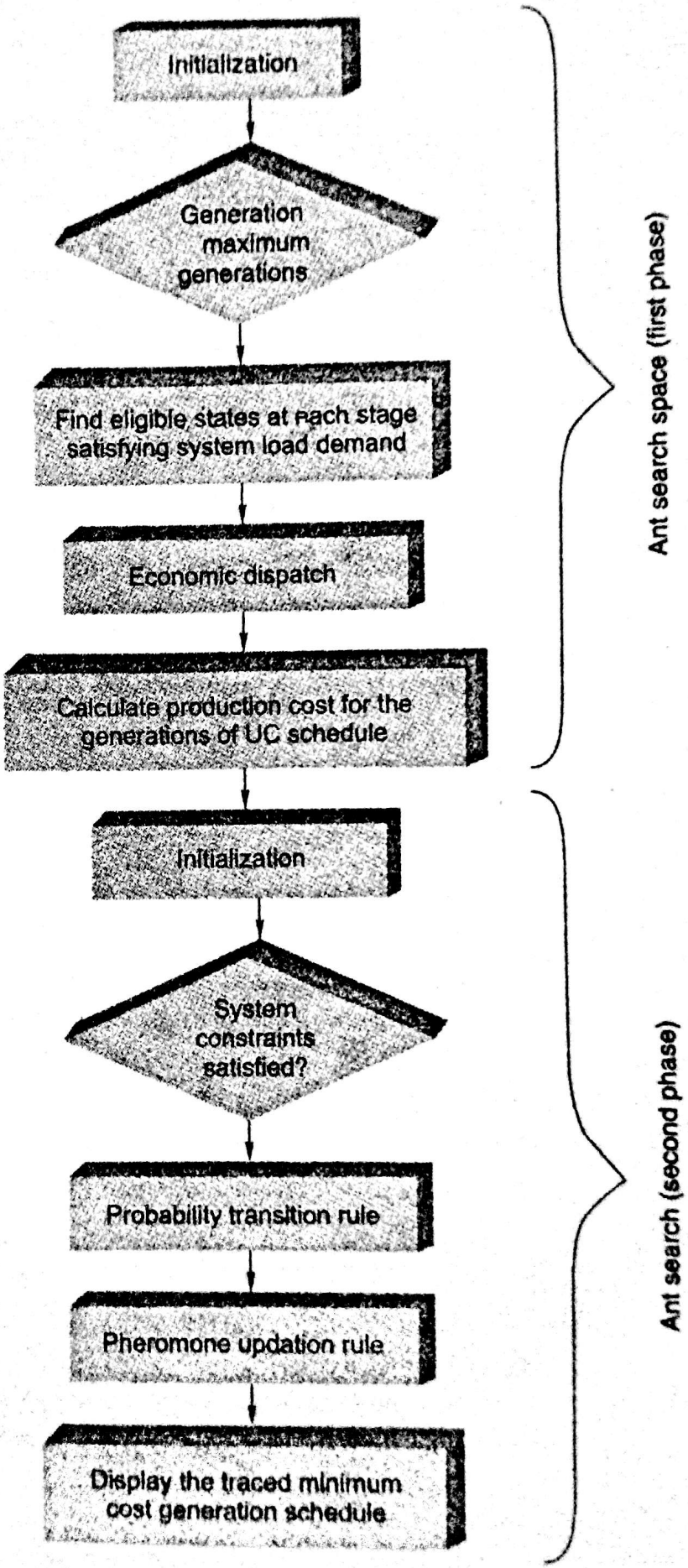


Fig. 10.9 ACS-based UCP flow chart

Table 10.6 Parameter settings for the ACS model (default settings: $\beta = 1$, $\rho = 0.3$, $\tau_0 = 1$, $q_0 = 0.7$)

β	Av. TGC ($/day)	ρ	Av. TGC ($/day)	q_0	Av. TGC ($/day)
0	27,324.25	0	27,568.67	0.1	27,937
1	27,224.5	**0.1**	**27,149.25**	0.3	27,685
2	27,206	0.2	27,170.25	0.5	2.7478
5	27,115	0.3	27,328	0.7	27,328
7	27,107.5	0.5	27,373.5	**0.9**	**27,135**
10	**27,030.25**	0.7	27,442.67		
15	27,151.5				

The results of the experiment show that in this model, a high value of α means that the trail is very important, and therefore ants tend to choose edges chosen by other ants in the past. This is true until the value of β becomes very high: in this case even if there is high amount of pheromone on a trail, there is always a high probability of an ant choosing another trail of lower cost. The ACS model performs well when the tuning factor is set to a high value, indicating that the model incorporates higher exploitation of the knowledge available about the problem. High values of β and/or low values of α make the algorithm very similar to a stochastic *multigreedy* algorithm.

The evolution of the convergence of the minimum cost path can be seen from Fig. 10.10—after the minimum number of ants have passed through the ant search space and completed their tours, the convergence of the cost finally becomes stagnant. Even after the stagnant situation, ants still try to explore any of the minimum cost paths that are available, which indicates the strength of the ant colony system model. This can be understood from Fig. 10.11 by the oscillations for each individual ant taking its tour.

The test results indicate that an ant can absorb information from the experience gained from its fellow ants' behaviour to get the best results, which are almost globally optimum. Even though at one transition stage, the cost of the ACS approach may be high as compared to dynamic programming (DP) approach, there is an overall reduction of generation cost because the ants can still foresee and pick out the stages coming afterwards which can belong to a low-cost path. It can be also seen from the literature that the DP approach is even now considered the best choice; however, the above results show that the ACS approach is performing successfully and helps to reduce the TGC more efficiently. The TGC for the four-unit system obtained using the DP approach is 27,541 $/day. Therefore, a saving of 2.2367% is obtained using the ACS model. A comparison of the transitional cost for the conventional dynamic programming and the ACS approach can be observed in Fig. 10.12 and Table 10.7. The hardware platform that has been used to implement this model is AMD Athlon[tm] XP 1800 + 1.54 GHz, 224 MB of RAM using the MATLAB environment.

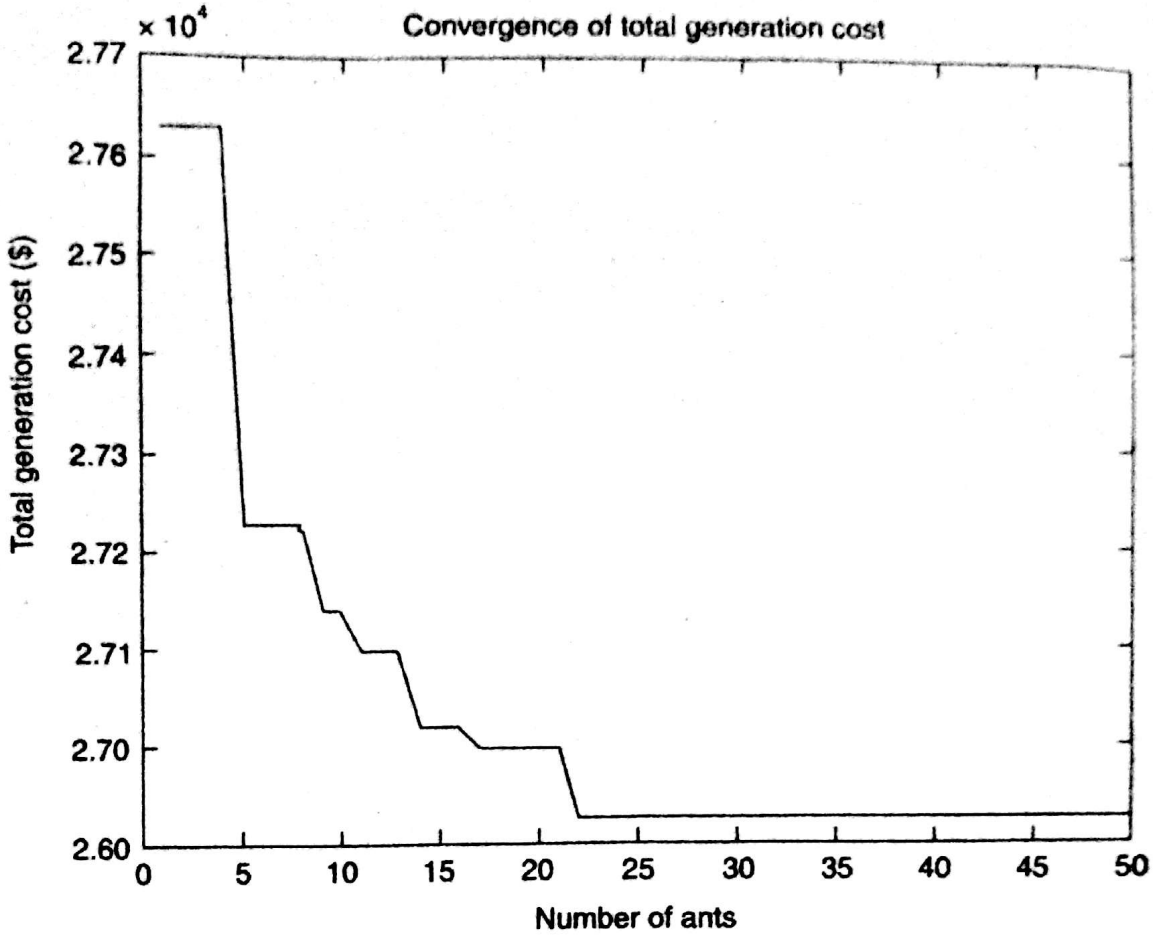


Fig. 10.10 Convergence of total generation cost

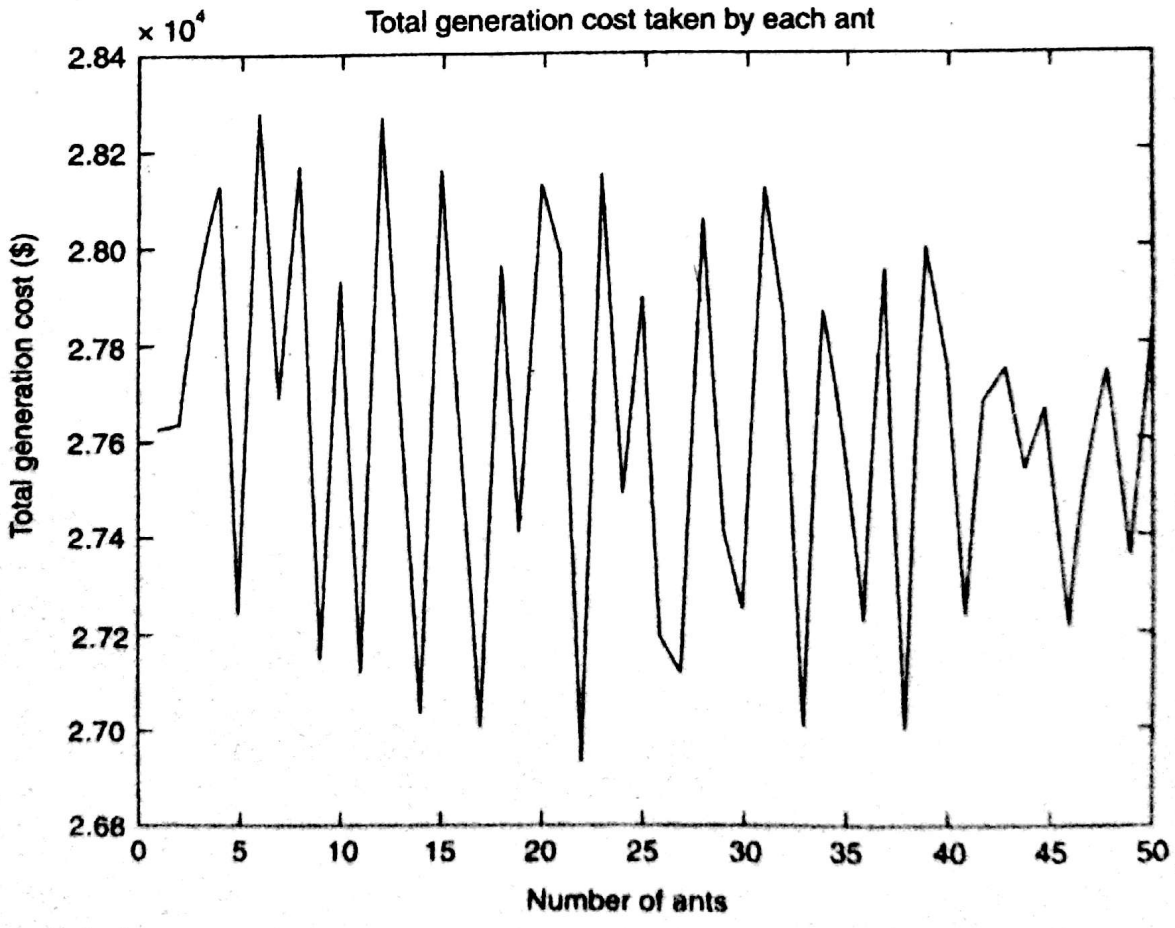


Fig. 10.11 Total generation cost path taken by each ant

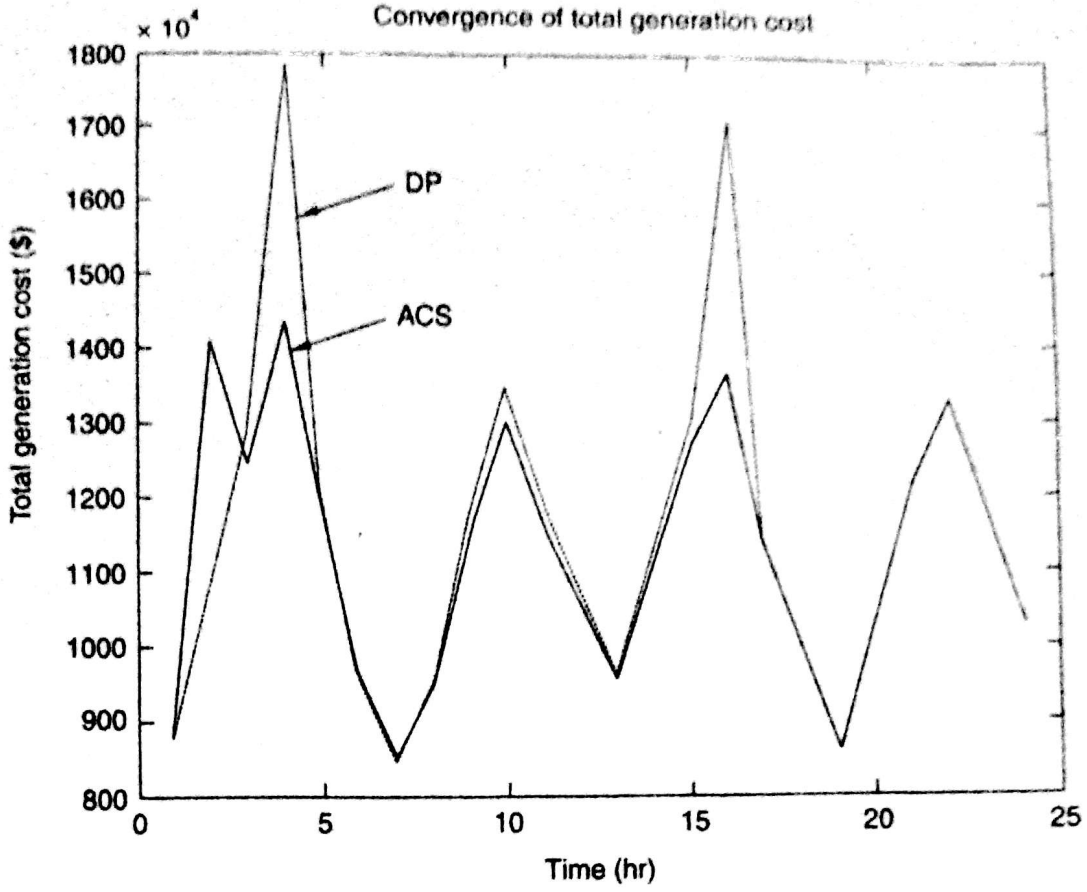


Fig. 10.12 Comparison of generation cost

Table 10.7 Transitional cost (TC)

Time period (hr)	*Demand (MW)*	*Dynamic programming*		*Ant colony system*	
		Unit status	*TC ($/hr)*	*Unit status*	*TC ($/hr)*
1	410	0111	0864.30	0111	0864.30
2	500	0111	1079.60	1111	1407.50
3	575	0111	1277.00	1111	1237.40
4	650	1111	1783.40	1111	1433.40
5	555	1111	1187.90	1111	1187.90
6	450	0111	0957.30	1110	0963.20
7	400	0111	0841.70	1110	0850.40
8	445	0111	0945.40	1111	0935.30
9	535	0111	1169.40	1111	1139.30
10	600	0111	1348.70	1111	1300.90
11	540	0111	1182.50	1111	1151.50
12	495	0111	1067.10	1111	1046.00
13	450	0111	0957.30	1111	0946.10
14	516	0111	1120.30	1111	1094.60
15	585	0111	1305.20	1111	1262.60
16	625	1111	1716.00	1111	1366.00

(contd)

Table 10.7 *(contd)*

Time period (hr)	*Demand (MW)*	*Dynamic programming*		*Ant colony system*	
		Unit status	*TC ($/hr)*	*Unit status*	*TC ($/hr)*
17	530	1111	1127.60	1111	1127.60
18	465	1111	0978.80	1111	0978.80
19	405	1111	0851.20	1111	0851.20
20	492	1111	1039.20	1111	1039.20
21	568	1111	1220.00	1111	1220.00
22	610	1111	1326.70	1111	1326.70
23	550	1111	1175.70	1111	1175.70
24	483	1111	1018.80	1111	1018.80
Total cost ($/day)			27,541		26,925
Time taken (s)			5.875		143.9840

In this section, a new unit commitment schedule has been presented using the ant algorithm approach. Since ant algorithms are more suitable for combinatorial optimization problems and have the potential of finding nearly global optimum solutions, they are very well suited for solving the unit commitment problem. Along with determining the minimum cost path, other related features based on pheromone deposition, such as pheromone updation and optimal control parameters (e.g., the tuning factor and relative importance of the trail), have been discussed.

10.8 PARTICLE SWARM INTELLIGENT SYSTEMS

The origin of the particle swarm intelligent system (PSIS) dates back to the year 1995. It was developed by Dr Eberhart and Dr Kennedy as an optimization technique known as particle swarm optimization (PSO) inspired by the flocking of birds. The concept is simple, has few parameters, is easy to implement, and has found applications in many areas. This intelligent technique has been researched extensively, and scientists are exploring its potential as an optimizer applicable to many fields of engineering. The PSIS originated as a simulation of a simplified social system. The main idea was to simulate the unpredictable choreography of a bird flock. These simulations were analysed to incorporate nearest-neighbour velocity matching, eliminate ancillary variables, and incorporate multidimensional search and acceleration by distance. Based on the observation of the evolution of the algorithm, it has been realized that the conceptual model is in fact an optimizer.

PSO can be categorized into five parts: algorithms, topology, parameters, merging or combining with other evolutionary techniques, and applications. Initially, PSO was developed for real-valued problems; however, it can be extended to cover binary and discrete problems. Its most exciting industrial application has been ingredient mix optimization by a major American corporation. In this work, 'ingredient mix' refers to the mixture of ingredients that are used to grow production

strains of microorganisms. The PSO provided an optimized ingredient mix that has over twice the fitness value found using traditional methods, at a very different location in ingredient space. The occurrence of an ingredient becoming contaminated hampered the search for a few iterations, but in the end did not give poor results; PSO is thus considered robust. PSO by nature searches a much larger portion of the problem space than the traditional method. It was used for reactive power and voltage control by a Japanese electric utility; it was employed to find a control strategy with continuous and discrete control variables, resulting in a sort of hybrid binary and real-valued version of the algorithm. Voltage stability in the system was achieved using a continuous power flow technique.

Particle swarm optimization has also proved its efficiency in evolving neural networks, i.e., training neural networks using particle swarms. So, like the other evolutionary computation algorithms, PSO can be applied to solve most optimization problems as well as problems that can be converted into optimization problems. It has been successfully applied for tracking dynamic systems and tackling multi-objective optimization and constraint optimization problems. The potential application areas also include classification, pattern recognition, biological system modelling, scheduling (planning), signal processing, games, robotic applications, decision-making, and simulation and identification. Examples include fuzzy controller design, job shop scheduling, real-time robot path planning, image segmentation, EEG signal simulation, speaker verification, time frequency analysis, modelling the spread of antibiotic resistance, burn diagnosing, gesture recognition, automatic target detection, etc. This natural phenomenon has thus proved successful and has paved the way for future research (Eberhart & Shi 2001).

10.8.1 The Basic PSO Method

PSO is initialized by a population of random solutions and each potential solution is assigned a randomized velocity. The potential solutions, called *particles*, are then 'flown' through the problem space. Each particle keeps track of its coordinates in the problem space, which are associated with the best solution or fitness achieved so far. The fitness value is also stored. This value is called *pbest*. Another 'best' value that is tracked by the global version of the particle swarm optimizer is the overall best value, and its location, obtained so far by any particle in the population. This value is termed *gbest*. Thus, at each time step, the particle changes its velocity (accelerates) and moves towards its pbest and gbest; this is the global version of PSO. When, in addition to pbest, each particle keeps track of the best solution, called *nbest* (neighbourhood best) or *lbest* (local best), attained within a local topological neighbourhood of the particles, the process is known as the local version of PSO. In addition, with respect to different applications, the discrete or binary version of PSO has come into existence. This is due to applications like scheduling or routing problems, for which some changes have to be made in order to adapt to discrete spaces (Eberhart & Kennedy 1995; Eberhart & Shi 2001).

10.8.2 Characteristic Features

The PSO method appears to adhere to the five basic principles of swarm intelligence (Parsopoulos & Vrahatis 2002).

(a) Proximity—the swarm must be able to perform simple space and time computations.
(b) Quality—the swarm should be able to respond to quality factors in the environment.
(c) Diverse response—the swarm should not commit its activities along excessively narrow channels.
(d) Stability—the swarm should not change its behaviour every time the environment alters.
(e) Adaptability—the swarm must be able to change its behaviour, when the computational cost is not prohibitive.

Indeed, the swarm in PSO performs space calculations for several time steps. It responds to the quality factors implied by each particle's best position and the best particle in the swarm, allocating the responses in a way that ensures diversity. Moreover, the swarm alters its behaviour (state) only when the best particle in the swarm (or in the neighbourhood, in the local variant of PSO) changes. Thus, it is both adaptive and stable (Eberhart et al. 1996).

10.8.3 Procedure of the Global Version

The algorithm of particle swarm optimization is as follows.

(1) Initialize an array of the population of particles with random positions and velocities in D dimensions in the problem space.
(2) Evaluate the fitness function in D variables for each particle.
(3) Compare each particle's fitness evaluation with its pbest. If the current value is better than pbest, then save the current value as pbest and let its location correspond to the current location in D-dimensional space.
(4) Compare the fitness evaluation with the population's overall previous best. If the current value is better than gbest, then save the current value as gbest to the current particle's array index and value.
(5) Modify the velocity and position of the particle according to the following equations:

$$v_{id}^{t+1} = v_{id}^{t} + c_1 \text{rand}()^t \times (p_{id}^{t} - x_{id}^{t}) + c_2 \text{Rand}()^t \times (P_{gd}^{t} - x_{id}^{t}) \quad (10.23)$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \quad (10.24)$$

(6) If the desired criterion is not met, go to step (2), otherwise stop the process. Usually the desired criterion may be a good fitness function or a maximum number of iterations.

Suppose the search space is D-dimensional, then the ith particle of the swarm can be represented by a D-dimensional vector $X_{id} = (x_{i1}, x_{i2}, ..., x_{iD})^T$. The velocity (position change) of this particle can be represented by another D-dimensional vector $V_{id} = (v_{i1}, v_{i2}, ..., v_{iD})^T$. The best previously visited position of the ith particle is denoted as $P_{id} = (p_{i1}, p_{i2}, ..., p_{iD})^T$. Defining g as the index of the best particle in the swarm (i.e., the gth particle is the best) and letting the superscripts denote the iteration number, the swarm is manipulated according to Eqns (10.23) and (10.24), where $d = 1, 2, ..., D$, $i = 1, 2, ..., N$, N is the size of the swarm, c is a positive constant called the acceleration constant, rand() and Rand() are random numbers uniformly distributed in [0, 1], and t determines the iteration number. The algorithm given above with Eqns (10.23) and (10.24) is the basic version of PSO. However, this version has no mechanism to control the velocity of the particle, which compels one to impose the maximum values V_{max} in the positive direction and $-V_{max}$ in the negative direction. This can be explained through the following equations:

$$\text{If } v_{id} > V_{max}, \text{ then } v_{id} = V_{max} \tag{10.25}$$

$$\text{If } v_{id} < -V_{max}, \text{ then } v_{id} = -V_{max} \tag{10.26}$$

This parameter proves to be very critical, because large values could result in particles moving away from good solutions, while small values result in inefficient exploration of the search space. This lack of a control mechanism for the position velocity results in the poorer performance of the PSO when compared to other evolutionary computation (EC) techniques. In particular, PSO is able to locate the optimum area faster than EC techniques, but fails in adjusting its velocity step size to continue the search for a finer grain. To overcome this limitation, the problem is addressed by incorporating a weight parameter for the previous velocity of the particle. The modified version of the equations is

$$v_{id}^{t+1} = \psi(wv_{id}^t + c_1\text{rand}()^t(p_{id}^t - x_{id}^t) + c_2\text{Rand}()^t(p_{gd}^t - x_{id}^t)) \tag{10.27}$$

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1} \tag{10.28}$$

where w is the *inertia weight*, c_1 and c_2 are two positive constants called the *cognitive parameter* and *social parameter*, respectively, rand() and Rand() are two random numbers generated independently, and ψ is a constriction factor used alternatively to w to limit velocity (Angeline 1998; Eberhart & Shi 2001).

10.8.4 Parameter Setting

It becomes necessary to choose optimum values for the parameters for the best performance of PSO for different types of applications. So, the selection of the important parameters, such as (i) pbest (p_{id}), (ii) nbest (p_{nd}) and gbest (p_{gd}), (iii) learning factors (c_1, c_2), (iv) inertia weight (w), and (v) constriction factor (ψ), has to be taken care of, which is discussed in the following (Eberhart & Shi 2001; 2004; Hu et al. 2004).

pbest pbest (p_{id}) is the best position of the particle attained so far and can be considered as the particle's memory; one memory slot is allotted to each particle. The best location does not necessarily always depend on the value of the fitness function. To adapt to different problems, many constraints can be applied to the definition of the best location. In certain non-linear constrained optimization problems, the particles remember the positions in feasible space and disregard unfeasible solutions. This simple alteration successfully locates the optimum solution for a series of benchmark problems. In addition, in some multi-objective optimization (MO) problems, the best positions are determined by a concept called *Pareto dominance*, i.e., if solution A is not worse than solution B in every objective dimension and is better than solution B in at least one objective dimension, solution B is dominated by solution A. In some other techniques, the memory reset mechanism is adopted; i.e., in dynamic environments, the particle's pbest will be reset to the current value if the environment changes.

nbest and gbest The best position that the neighbours of a particle have achieved so far is nbest (p_{nd}), and gbest (p_{gd}) is the best of nbest; the whole population is taken as the neighbourhood of each particle. The neighbourhood of a particle is the social environment encountered by it. The selection of nbest consists of two phases. In the first phase the neighbourhood is determined and in the second phase nbest is selected. Usually, certain predetermined conjunct particles are considered as neighbours. Neighbours are defined as topological neighbours; neighbourhoods do not change during a run. The number of neighbours or the size of the neighbourhood affects the convergence speed of the algorithm. Larger the size of the neighbourhood, more the observed convergence rate of the particles. Premature convergence or pre-convergence of the particles is prevented by keeping the neighbourhood size small. Although various neighbourhood structures and their influences on performance have been studied, no conclusive results have been reached so far; this is an area of further research. nbest is usually selected by comparing fitness values among neighbours. If the neighbourhood size is defined as 2, for instance, particle i compares its fitness value with particle $i - 1$ and particle $i + 1$. According to the literature available and experience based on trial and error, usually a neighbourhood size of about 15% of the population size is used for many applications. So, for a population of 40 particles, a neighbourhood size of six or three topological neighbours on each side is generally taken. The population size is problem-dependent, and population sizes of 20 to 30 particles are probably most common. So, smaller populations are optimal for PSO in terms of minimizing the total number of evaluations (i.e., population times the number of generations) required to obtain a good solution. This will be difficult in the case of a multi-objective optimization environment, where multiple fitness values for each particle have to be taken care. In certain cases, the ratio of the fitness and the distance from other particles is used to determine nbest.

Learning factors The constants c_1 and c_2 are known as learning factors. They represent the weighting of the stochastic acceleration terms that pull each particle towards the pbest and nbest positions. Thus, adjustments of these constants change the amount of 'tension' in the system. Low values allow particles to roam far from target regions before being tugged back, while high values result in abrupt movements towards, or past, the target regions. The cognitive parameter represents the tendency of individuals to duplicate past behaviours that have proven successful, whereas the social parameter represents the tendency to follow the successes of others. Generally, c_1 and c_2 are set to 2.0, which will make the search cover all surrounding regions centred at pbest and nbest. Also, if the learning factors are identical, the same importance is given to social searching and cognitive searching (i.e., both parts are essential to the success of particle swarm searching).

Inertia weight Inertia weight w has become very important for the convergence behaviour of PSO. As already mentioned, the maximum velocity V_{max} is a constraint that controls the global exploration ability of a particle swarm. It has been understood that larger V_{max} facilitates global exploration, while smaller V_{max} encourages local exploitation. The concept of inertia weight has been developed to have a better control over exploration and exploitation. Shi and Eberhart first introduced this concept in the year 1998. A suitable value for the inertia weight w usually provides a balance between global and local exploration abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. In earlier solutions, the inertia weight was set to a constant initially, but later experimental results suggested having a larger value initially in order to promote global exploration of the search space, and gradually decreasing it to get more refined solutions. Thus, an initial value of around 1.2 and a gradual decline towards 0 can be considered a good choice for w. Also, randomized inertia weights have been used in many reports, i.e., the inertia weight can be set to [0.5+ rand()/2.0] according to Clerc's constriction factor (Clerc 1999).

Constriction factor The constriction factor ψ controls the magnitude of the velocities in a way similar to the parameter resulting in a variant of PSO, different from that with inertia weight. The work done by Clerc (1999) suggests that the use of ψ may be necessary to ensure convergence of PSO. The constriction factor is considered a function of the accelerating constants c_1 and c_2 as given by the following equation:

$$\psi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \tag{10.29}$$

where $\varphi = c_1 + c_2$, $\varphi > 4$. Here, according to the Clerc's constriction factor, φ is set to 4.1 and the constant multiplier ψ thus becomes 0.729. Even though initially it was thought that V_{max} is not necessary, from subsequent experimental results it has been found that V_{max} can be limited, and is often set at about 10–20% of the

dynamic range of the variable in each dimension, while also selecting an appropriate inertia weight w.

10.8.5 Comparison with Other EC Techniques

PSO is an evolutionary computation technique because it has the common evolutionary attributes as detailed below (Angeline 1998; Wang et al. 2003).

(a) During initialization, there is a population that is made up of a certain number of individuals, and each individual in the population is given a random solution initially.
(b) It has a mechanism for searching for a better solution in the problem space and producing a better new generation.
(c) The production of the new generation is based on the previous generation.

PSO can easily be implemented and is computationally inexpensive, since its memory and CPU speed requirements are low. Moreover, it does not require gradient information of the objective function under consideration, but only its values, and it uses only primitive mathematical operators. PSO has been proved to be an efficient method for many global optimization problems, and in some cases it does not suffer the difficulties encountered by other EC techniques.

In EC techniques, three main operators are involved: recombination, mutation, and selection. PSO does not have a direct recombination operator. However, the stochastic acceleration of a particle towards its previous best position as well as towards the best particle of the swarm (or towards the best particle in its neighbourhood in the local version) resembles the recombination procedure of EC. In PSO, information exchange takes place only between the particle's own experience and the experience of the best particle in the swarm, instead of being carried forward from 'parents' selected based on their fitness to descendants as in genetic algorithms (GAs).

Moreover, the PSO's directional position updating operation resembles the mutation of GAs, with a kind of in-built memory. This mutation-like procedure is multidirectional in PSO, like in GA, and includes control of the mutation's severity using factors such as V_{max} and ψ.

PSO is actually the only evolutionary algorithm that does not use the 'survival of the fittest' concept. It does not utilize a direct selection function. Thus, particles with lower fitness can survive during the optimization and potentially visit any point of the search space.

10.9 ENGINEERING APPLICATIONS OF PSIS AND FUTURE RESEARCH

PSO is attractive due to easy implementation and very few parameters to adjust and, therefore, has been used in a wide variety of applications. Many of PSO's applications have already been pointed out in the previous sections. As mentioned

before, it can be used in the place of other EC techniques. It is also being used nowadays for training artificial neural networks; not only the network weights but also the network structure is evolved. As an example of evolving neural networks, PSO has been applied to the analysis of human tremor.

PSO has become very popular and researchers are trying to apply it to various fields of engineering. However, it is necessary to know its scope in the near future. The still many unexplored areas of PSO are given below (Eberhart & Shi 2001; Hu et al. 2004).

(a) Convergent analysis: It is still not clear how PSO converges, so thorough work has to be done in the theoretical research of swarm intelligence and chaos systems.

(b) The combination of various PSO techniques as well as other hybridized techniques with PSO dealing with complex problems have to be understood.

(c) Discrete/binary PSO: Available literature has shown the potential of EC techniques in dealing with discrete or binary variables. However, in the case of PSO, some difficulties have been encountered, which have yet to be solved.

(d) PSO can be treated as an agent-based distributed computational technique; many of its computing characteristics still remain to be uncovered.

The travelling salesman problem The TSP is a well-known NP-hard combinatorial problem. There are very few papers in the literature of PSO applied to the TSP. This is in fact due to the difficulties faced in using PSO for solving discrete optimization problems. However, Kang-Ping Wang et al. (2003) tried to solve the TSP using the concept of a swap operator (SO) and the swap sequence (SS) mechanism. We again consider the symmetric TSP problem, a static combinatorial optimization problem which has N cities or nodes. Before applying PSO to the TSP, it is important to know the basic concepts of SO and SS.

Let us consider a general solution sequence of the TSP having N nodes, $S = (a_i)$, $i = 1, \ldots, N$. $\mathrm{SO}(i_1, i_2)$ can be defined as the exchange of node a_{i1} and node a_{i2} in the solution S. Therefore, the new solution $S' = S + \mathrm{SO}(i_1, i_2)$ is obtained when $\mathrm{SO}(i_1, i_2)$ acts on S, where '+' has its usual meaning. For example, let $S = (3, 6, 4, 1, 2, 5)$. Using SO(1, 2), we get

$$S' = S + \mathrm{SO}(1, 2) = (3, 6, 4, 1, 2, 5) + (1, 2) = (6, 3, 4, 1, 2, 5)$$

Now a swap sequence is made up of one or more SO's.

$$\mathrm{SS} = (\mathrm{SO}_1, \mathrm{SO}_2, \ldots, \mathrm{SO}_N) \tag{10.30}$$

Attributes of the swap sequence If a swap sequence is acting on a solution, then all the swap operators of that swap sequence act on the solution in an order given by the following formula:

$$\begin{aligned} S' &= S + \mathrm{SS} = S + (\mathrm{SO}_1, \mathrm{SO}_2, \ldots, \mathrm{SO}_N) \\ &= [\ldots, ([(S + \mathrm{SO}_1) + \mathrm{SO}_2] + \mathrm{SO}_3), \ldots, \mathrm{SO}_N] \end{aligned} \tag{10.31}$$

It may be observed that a different swap sequence acting on the same solution may produce the same new solution. All the swap sequences which give the same new solution are said to belong to the *equivalent set of swap sequences*. The swap sequence which has the least number of swap operators is called the basic swap sequence (BSS) of the set.

It is also possible to merge two swap sequences, producing a new swap sequence, defined by the operator $\oplus$. Let there be two swap sequences SS_1 and SS_2. The merging of SS_1 and SS_2 can be explained as SS_1 and SS_2 acting on the solution S in order. First SS_1 acts on S and a solution S_1 is obtained; then SS_2 acts on S_1 to get S', which is the new solution obtained. We must also know that there exists a swap sequence SS′, which, when operated on the same solution S, gives the same new solution S'. This can be understood from the following equations:

$$SS_1 \oplus SS_2 = S + (SS_1\ SS_2) = [(S + SS_1) + SS_2)]$$
$$= S_1 + SS_2 = S' \qquad (10.32)$$

Also,

$$S + SS' = S' \qquad (10.33)$$

Since SS′ and $SS_1 \oplus SS_2$ are in the same equivalent set,

$$SS' = SS_1 \oplus SS_2 \qquad (10.34)$$

Suppose there are two solutions A and B, a basic swap sequence SS that can act on B and give A has to be constructed. Defining $SS = A - B$, the nodes in B are swapped according to A from left to right to get SS. The minus sign (–) has now got its new meaning. So, the equation can be changed to $B + SS = A$. For example, let the two solutions be

$$A = (1, 2, 3, 4, 5)$$
$$B = (2, 3, 1, 5, 4)$$

Therefore,

$$A(1) = B(3) = 1$$

So, the first swap operator is SO(1, 3).

$$B_1 = B + SO(1, 3) = (1, 3, 2, 5, 4)$$

Now,

$$A(2) = B_1(3) = 2$$

So, the second swap operator is SO(2, 3).

$$B_2 = B_1 + SO(2, 3) = (1, 2, 3, 5, 4)$$

Now,

$$A(4) = B_2(5) = 2$$

So, the third swap operator is SO(4, 5).

$$B_3 = B_2 + \text{SO}(4, 5) = (1, 2, 3, 5, 4)$$

Now $B_3 = A$. Thus, we have obtained the basic swap sequence $\text{SS} = A - B = (\text{SO}(1, 3), \text{SO}(2, 3), \text{SO}(4, 5))$.

Merging operators used in PSO Equation (10.23) can be modified by incorporating the SS and SO operators:

$$v_{id} = v_{id} \oplus R_1(p_{id} - x_{id}) \oplus R_2(p_{gd} - x_{id}), \quad R_1, R_2 \in [0, 1] \tag{10.35}$$

where R_1 and R_2 are random numbers generated between 0 and 1. The term $R_1(p_{id} - x_{id})$ means that all swap operators in the basic swap sequence $(p_{id} - x_{id})$ are maintained with a probability of R_1, which is the same as for $R_2(p_{gd} - x_{id})$. This indicates that the bigger values of R_1 have a greater influence on p_{id} for more swap operators in $(p_{id} - x_{id})$, which is also the same as in the case of R_2 influencing p_{gd} for more swap operators in $(p_{gd} - x_{id})$.

Step-by-step algorithm for TSP using PSO

(1) Initialize the array of population of particles with their random solutions (positions) and a random swap sequence (velocity).
(2) Evaluate the next position x'_{id} for all the particles x_{id}.
   (a) Calculate the difference between p_{id} and x_{id} according to the concept of the BSS. Here $A = p_{id} - x_{id}$, which is the BSS.
   (b) Calculate $B = p_{gd} - x_{id}$, which is also a BSS.
   (c) Calculate the velocity v_{id} according to Eqn (10.35), where the swap sequence v_{id} has to be transformed into a BSS.
   (d) Calculate the new solution

$$x_{id} = x_{id} + v_{id} \tag{10.36}$$

   Here the swap sequence acts on the solution x_{id} to get a new solution.
(3) Update p_{id} if the new solution is superior to p_{id}.
(4) Update p_{gd} if the new best solution is superior to p_{gd}.
(5) If the desired criterion is not met, go to step (2); otherwise stop the process. Here the desired criterion is the shortest distance covered by the travelling salesman in visiting all the cities and finally reaching the initial position or a required number of iterations to obtain the shortest distance travelled by the salesman.

Consider the TSP with 14 nodes as shown in Table 10.8. Assume that the number of particles chosen is 100, the maximum number of iterations is 20,000, the size of the search space is 2,000,000, and the solution space is 0.064%. The optimal solution of this TSP is found to be 34.2805 units and the node path is 1-10-9-11-8-13-7-12-6-5-4-3-14-2-1 (Fig. 10.13). Even though there may be many methods yet to be discovered to solve the TSP using PSO, the above procedure will give an idea about the possible applications of PSO.