PAPER

# A New Local Search Based Ant Colony Optimization Algorithm for Solving Combinatorial Optimization Problems

**Md. Rakib HASSAN**[†], **Md. Monirul ISLAM**[†a)], ***Nonmembers***, ***and*** **Kazuyuki MURASE**[†], ***Member***

**SUMMARY** Ant Colony Optimization (ACO) algorithms are a new branch of swarm intelligence. They have been applied to solve different combinatorial optimization problems successfully. Their performance is very promising when they solve small problem instances. However, the algorithms' time complexity increase and solution quality decrease for large problem instances. So, it is crucial to reduce the time requirement and at the same time to increase the solution quality for solving large combinatorial optimization problems by the ACO algorithms. This paper introduces a Local Search based ACO algorithm (LSACO), a new algorithm to solve large combinatorial optimization problems. The basis of LSACO is to apply an adaptive local search method to improve the solution quality. This local search automatically determines the number of edges to exchange during the execution of the algorithm. LSACO also applies pheromone updating rule and constructs solutions in a new way so as to decrease the convergence time. The performance of LSACO has been evaluated on a number of benchmark combinatorial optimization problems and results are compared with several existing ACO algorithms. Experimental results show that LSACO is able to produce good quality solutions with a higher rate of convergence for most of the problems.

***key words:*** *ant colony optimization, combinatorial optimization problem, local search*

## 1. Introduction

Ant Colony Optimization (ACO) algorithms [1] were introduced by M. Dorigo and colleagues in the early 1990s as a novel nature-inspired metaheuristic for solving difficult combinatorial optimization problems [1] in a reasonable amount of time. These algorithms use a number of artificial ants with a set of simple rules that take inspiration from the behavior of real ants for solving a given problem. Artificial ants are allowed to move freely on a graph, which represents a given problem being solved. These ants probabilistically build a solution to the problem and then deposit some artificial pheromone on the edges of the graph to bias the solution construction activity of future ants. The amount of pheromone deposition and the way of building solutions are such that the overall search process is biased towards better approximate solutions. ACO algorithms have been applied to many combinatorial optimization problems such as traveling salesman problem [3], job-shop scheduling [4], sequential ordering problem [5], resource constraint project scheduling problem [6] and open shop scheduling problem [7].

There exist several variants of ACO algorithms. These are ant system (AS) [8], elitist AS [9], ant-Q [10], [11], ant colony system (ACS) [12], max-min AS [13], rank-based AS [14], approximate nondeterministic tree search [15], [16], and hyper-cube AS [17], [18]. We briefly describe these algorithms in Sect. 2. The performance of these ACO algorithms in solving difficult combinatorial optimization problems is very promising [19]. However, the solution quality of these algorithms decreases as the size of problems grows [20]–[24]. These algorithms stuck in local optima or take much time to solve large problem instances.

In this paper, we propose a new ACO algorithm, called Local Search based ACO algorithm (LSACO). Our LSACO differs from the existing ACO algorithms in different ways. It uses a new adaptive local search to find optimum solutions, so the name is local search based ACO. The proposed algorithm balances the exploration and exploitation of the search space in such a way that the chance of getting stuck into local optima is greatly reduced. The pheromone initialization and updating is done in different ways in LSACO. To reduce the time complexity, LSACO adopts a number of different techniques. These are the use of don't look bit, dynamic ordering of the candidates, and a maximum of 5-opt move.

The rest of the paper is organized as follows. Section 2 discusses the related works of ACO and their potential problems. Section 3 describes LSACO in detail and gives motivations behind various design choices and ideas. Section 4 presents experimental results on LSACO and some discussions. Finally, Sect. 5 concludes with a summary of this paper and few remarks.

## 2. Related Work

A number of ACO algorithms have been proposed in the literature. All such algorithms use two components: the artificial ants and their deposited pheromone. They mainly differ in some aspects of the search control. The first ACO algorithm is the AS [8], which was applied to solve the traveling salesman problem. The relative performance of AS tends to decrease dramatically when the size of the test-instance increases.

A first improvement on the initial AS is the Elitist AS [9]. The idea of Elitist AS is to provide strong additional reinforcement to the edges belonging to the so-far best tour found. The performance of Elitist AS is not good compared to ACS [12]. Rank-based AS ($AS_{rank}$) [14], another improvement over AS, assigns ranks to the ants based

Manuscript received March 30, 2009.
Manuscript revised October 24, 2009.

[†]The authors are with Fukui University, Fukui-shi, 910–8507 Japan.
a) E-mail: monirul@synapse.his.fukui-u.ac.jp

DOI: 10.1587/transinf.E93.D.1127

on their performance. The quantity of pheromone deposition by an ant is weighted according to its rank. $AS_{rank}$ performs slightly better than the Elitist AS with an extra overhead due to ranking. Ant-Q [10], [11] is developed based on AS and Q-learning algorithms. Although its performance is good, Ant-Q is abandoned for the equally good but simpler ACS [12].

Max-Min AS (MMAS) [13] introduces a number of modifications into AS. MMAS initializes the pheromone trails to the upper limit, restricts the possible range of pheromone trail values to avoid local optima, exploits the best tours found and reinitializes the pheromone trails each time the system approaches stagnation. Among the variant of ACO algorithms, ACS [12] is the best performing one that exploits search experience through an aggressive action choice rule. ACS deposits and evaporates pheromone only on the edges belonging to the best-so-far tour. When an ant moves from the node $i$ to the node $j$, ACS removes some pheromone from the edge in order to increase the exploration of alternative paths.

Approximate Nondeterministic Tree Search (ANTS) [15], [16] is an ACO algorithm that exploits ideas from mathematical programming. This algorithm introduces hyper-cube framework in ACO to rescale the pheromone value in the interval $[0, 1]$. The hyper-cube framework is a complex method compared to the existing ACO algorithms and so, it has not been applied extensively like other ACO algorithms.

Our proposed LSACO relates to the hyper-heuristics approaches [31]–[34], which broadly describe the process of using meta-heuristics to choose meta-heuristics for solving the problem in hand [31]. A hyper-heuristic does not operate on problems directly and does not utilize domain knowledge; rather it utilizes low level heuristics in solving problems. The aim of hyper-heuristics is to raise the level of generality on which search algorithms operate. Hyper-heuristics can be split into two groups [34]: constructive and perturbative (local search). Constructive hyper-heuristics construct a solution from scratch by using constructive heuristics. Perturbative hyper-heuristics start from a complete initial solution and iteratively select appropriate low level heuristics aiming at improving the solution. The LSACO algorithm falls in constructive hyper-heuristics.

For solving TSP problems, several other local search algorithms exist among which Lin-Kernighan heuristic [35], [36] performs best. Our proposed LSACO is different from Lin-Kernighan heuristic with respect to the base algorithm. LSACO uses a variant of ACO algorithms as its base algorithm in solving combinatorial optimization problems. This base algorithm is different from the one used in [35], [36]. The detail description of LSACO is given in the next section.

## 3. Proposed LSACO Algorithm

The LSACO algorithm consists of two main parts: base algorithm and adaptive local search. The base algorithm used in LSACO is a variant of ACS and consisted of three components: tour construction, local pheromone update rule, and global pheromone update rule. Thus there are four components in LSACO, three for the base algorithm and one for the adaptive local search. The LSACO algorithm executes its four components sequentially one after another and repeats the whole process again and again until a solution is found or the termination criterion is satisfied.

To improve the solution quality and reduce the computational complexity, a number of different techniques are adopted in LSACO. We describe different components and techniques in the following subsections.

### 3.1 Tour Construction

The LSACO first maps a given problem to a graph with a set of nodes and edges. The aim of LSACO is to find a minimum cost path in the graph, which is the solution of the problem. Our LSACO constructs tours in a different but effective way to improve the solution quality in a reasonable time. Initially, the algorithm places $m$ ants on the randomly chosen nodes of the graph, where $m$ is a user-specified parameter and selected randomly between 10 and 15. If the solution quality does not improve after every $\tau$ iteration, it is assumed that the number of ants is insufficient. Hence LSACO adds one ant and places the new ant on a new node, which is chosen randomly from the nodes that do not contain any ant. The number of iterations, $\tau$, is a user-specified parameter. The LSACO algorithm tests this criterion after every $\tau$ iterations. We can describe the criterion as:

$$m_{t+1} = \begin{cases} m_t + 1 & \text{if } (m_t < m_{\max}) \text{ and } Q(t) > Q(t-1) \\ m_t & \text{otherwise} \end{cases} \tag{1}$$

Here, $m_t$ is the number of ants at iteration $t$ and $m_{\max}$ is a user-specified parameter that determines the maximum number of ants. The variables $Q(t)$ and $Q(t-1)$ represent the solution quality at addition steps $t$, and $t-1$, respectively. We set $m_{\max}$ to 1.5 times of the initial number of ants. As execution proceeds, if LSACO finds that the addition of ants improves solution quality, it increases $m_t$ by one. This is done by comparing $m_t$ with $m_{\max}$ and $Q(t)$ with $Q(t-1)$ as expressed by Eq. (1). It is worth mentioning that the LSACO algorithm does not delete any ant even if it exhibits worse performance compared to other ants. This is reasonable in the sense that the worse ant may perform better in the future iterations due to the use of a variable-opt local search strategy in our algorithm.

To reduce burden in defining $m_{\max}$, LSACO uses an criterion to increase the value of $m_{\max}$ if it is set small. We can describe the criterion as:

$$m_{\max} = \begin{cases} m_{\max} + 1 & \text{if } (m_t = m_{\max}) \text{ and } T_e < T_a \\ m_{\max} & \text{otherwise} \end{cases} \tag{2}$$

where $T_e$ and $T_a$ represent amount of time already elapsed and amount of time allowed for solving a given problem,

respectively. At every iteration, our LSACO first checks the criterion expressed by Eq. (2) to add an ant according the criterion expressed by Eq. (1).

The way LSACO constructs tours is different from the existing ACO algorithms [8]–[18]. Generally, a fixed number of ants are used throughout the whole execution of an algorithm. The choice of this number is very crucial for the performance of an ACO algorithm. A large number of ants may increase computational expense, while a small number of ants may reduce the solution quality. To increase solution quality, AS [8] sets the number of ants equals to the number of nodes in a graph and keeps it fixed during execution. Although this is reasonable for a medium sized problem, it becomes infeasible when the problem size grows resulting in the increase of the nodes in a graph.

Since LSACO is a variant of the ACO algorithm, it uses a pheromone based transition rule that decides how ants move from one node to another based on pheromone in the edges of a graph. Our algorithm initializes each edge $(i, j)$ with an amount of pheromone $1/C^{nn}$, where $C^{nn}$ is the length of a nearest-neighbor tour of a problem size $n$. This initial pheromone value is neither too low nor too high. Thus, at the start of search, all edges will be good candidates for moderate exploration and exploitation. The positive effect of such an initialization is that it avoids early entrapping at local optima and low convergence in finding solutions. It has been known that the pheromone initialization plays an important role in the solution quality [25].

Existing ACO algorithms initialize the edges with a maximum or minimum pheromone level to encourage more exploration or exploitation at the beginning of search [8], [12]–[14]. Although this kind of extreme initialization may be suitable for some problems, it may not be suitable for other problems. For example, more exploration is suitable if a search space is very rough consisting of many local optima, while more exploitation is suitable for a smooth search space. Since it would not be possible to know the characteristics of search spaces in advance, the LSACO algorithm initializes pheromone with a moderate value. This initialization approach can be considered as a combination of two extremes: initialize pheromone with a very large value and initialize pheromone with a very small value.

According to a node transition rule, ants choose the next node $j$ from the unvisited nodes. This rule can be expressed as follows [12]:

$$j = \begin{cases} \arg\max_{u \in N_i^k} \{\tau_{iu}[\eta_{iu}]^\beta\}, & \text{if } q \le q_0 \text{ (exploitation)} \\ J, & \text{otherwise (biased exploration)} \end{cases} \tag{3}$$

where $N_i^k$ is the feasible neighborhood of the ant $k$ while being at node $i$, i.e., the set of nodes that the ant $k$ has not visited yet, $q$ is a random variable uniformly distributed in $[0, 1]$. According to [12], $q_0$ is set to 0.9, which is found optimal independent of problems. Every time an ant in node $i$ intends to move to a node $j$, the LSACO generates a random number $q$ uniformly in the range $[0, 1]$. If $q > q_0$, then the next node $j$ (denoted as $J$) is selected using Eq. (3). $J$ is a random variable selected according to the probability distribution and expressed by the following equation [8]:

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{u \in N_i^k} [\tau_{iu}]^\alpha [\eta_{iu}]^\beta}, & \text{if } j \in N_i^k \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Here, $p_{ij}^k$ is the probability of choosing the node $j$ from the node $i$ for the ant $k$, $\tau_{ij}$ is the pheromone trail of the edge $(i, j)$, and $\eta_{ij} = 1/d_{ij}$ is a heuristic in which $d_{ij}$ is the cost of the edge $(i, j)$. The parameter, $\alpha$ and $\beta$, determine the relative influence of the pheromone trail and the heuristic value, respectively. The aforementioned rule indicates that the probability of choosing a particular edge $(i, j)$ increases with the value of the associated pheromone trail $\tau_{ij}$ and of the heuristic information $\eta_{ij}$.

The value of $\alpha = 0$ means that the closest nodes are more likely to be selected. And $\beta = 0$ means that only pheromone value will be considered without any heuristic bias, that is, the distance of the nodes will not be considered. Thus $\alpha = 0$ or $\beta = 0$ will lead the solution to local optima. Existing ACO algorithms used $\alpha = 1$ to increase the chance of convergence by giving more importance to pheromone. This operation is equivalent to exploitation which is expressed by our Eq. (3). We used $\alpha = 0.1$ in Eq. (4) to decrease the chance of convergence by giving less importance to pheromone. Thus the shorter edges with more pheromone will be not frequently chosen by ants. This operation is equivalent to exploration. These phenomena indicate that unlike existing ACO algorithms, LSACO incorporates both exploitation and exploration in searching.

The tour construction phase is the major component of ACO algorithms because it affects the solution quality and convergence speed. As an approach to fasten the search process, the LSACO algorithm uses a candidate list, a list of preferred nodes to be visited, in the tour construction phase for problems larger than 1000 nodes. The candidate list is pre-computed by the nearest neighbor heuristic and kept same throughout the execution of our algorithm. For problems with more than 1000 nodes, the size of the candidate list is 50. The nodes in the candidate list are arranged in ascending order of distances. The ants in LSACO first consider the nodes belonging to the candidate lists to move to. If all nodes in the candidate list have already been visited then LSACO considers the rest of the unvisited nodes. The use of the candidate list reduces the time complexity nearly to $O(n)$, which without the candidate list is $O(n^2)$ where $n$ is the number of nodes. This reduction is important for solving large problems in reasonable time.

### 3.2 New Local Pheromone Update

As mentioned before, LSACO initializes the edges of a graph with pheromone and ants chose to move from one node $i$ to another node $j$ based on the probabilistic transition rule. It is natural to reduce the pheromone deposited

in the edge $(i, j)$ to undermine its desirability so that all the ants do not follow the same path. This phase is called as local pheromone update. Unlike AS [8], LSACO reduces pheromone only on the visited edges. The local pheromone updating rule of LSACO is as follows:

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} \tag{5}$$

where $\xi$, $0 < \xi < 1$, is the pheromone decay parameter. Since LSACO reduces pheromone only for the visited edges, its time complexity is linear i.e., $O(n)$. The time complexity of some other ACO algorithms (e.g. AS [8]) is $O(n^2)$, which is not suitable for large sized problems. After local pheromone updating, LSACO applies a new adaptive local search phase, which is described as follows.

### 3.3 Adaptive Local Search

Local search is an iterative method that works in ACO algorithms by changing the edges of a graph to further improve the solutions found by the ants. Generally, ACO algorithms employ 2-opt and 3-opt local search methods, a special case of the $\lambda$-opt algorithm [4], [5], [13], where $\lambda$ edges of the current tour are replaced in each step by the same number of edges in such a way that a shorter tour is achieved. Although any value of $\lambda$ can be used, existing ACO algorithms use a fixed value 2 or 3 for all problems. This fixed value may be suitable for some problems, but it may not be suitable for other problems and may degrade the solution quality. Figure 1 illustrates a sample edge exchanging process. Suppose the graph shown in the figure is a tour with minimum cost. Then at each iteration step the searching attempts to find two sets of links, $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ in such a way if the links of $X$ are replaced by the links of $Y$, then the result is a better tour.

To overcome the aforementioned problem, LSACO uses an adaptive approach in changing the value of $\lambda$ during execution. For the local search to take place in LSACO, a feasible tour with minimum cost $T_{min}$ is chosen. Given $T_{min}$, the adaptive approach repeatedly exchanges edges that reduce the length of the current tour. The exchanging process stops when no improvement is obtained by exchanging edges. Our adaptive local search starts with $\lambda = 2$ and then it examines $\lambda = \lambda + 1$. If $\lambda + 1$ yields a better solution, then $\lambda + 2$ is examined. This process is continued for $\lambda = 5$. We have found that $\lambda > 5$ does not improve the solution quality although it increases time complexity. This choice is not an inherent constraint because we allow $\lambda$ to change during execution. If a large value is found suitable for some problems, LSACO can use it because the adaptive nature of the proposed local search. Because of the adaptive nature of the local search, backtracking is not required in LSACO, which is necessary for existing ACO algorithms [8]–[18]. The avoidance of backtracking simplifies LSACO and tends to reduce runtime in solving difficult problems.

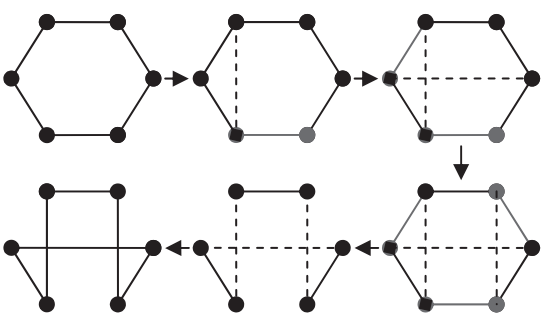

**Fig. 1** Edge exchanging process.

The search for better tours is stopped when the current tour is similar to a previous tour or when all the alternatives are searched. Every resulting final tour after exchanging edges must be better than the original one. Otherwise, the original tour is not replaced. To reduce the number of comparisons, thus to reduce time complexity, and to improve the performance of the algorithm, several measures have been taken in the local search of LSACO. The major new steps that are incorporated in this new algorithm are given in the following subsections.

#### 3.3.1 Don't Look Bit

To speed up the computation time, the concept of don't look bit [26] is used here. Suppose, $(i, j)$ is the edge connecting the nodes $i$ and $j$. Starting from $i$, there may be several edges to reach $j$. If the algorithm previously failed to find an improvement from node $i$, and node $i$'s tour neighbors have not changed since then, it is unlikely that an improvement can be made if the algorithm again looks at node $i$.

To incorporate this technique, LSACO uses don't look bit for each node, which is initially set to 0. The don't look bit for node $i$ is set to 1 whenever a search failed previously from node $i$, and it is set to 0 whenever an improving move is found. To consider candidates for node $i$, all nodes whose don't look bit is 1 are ignored. This is done by maintaining a queue of nodes whose bits are zero. The incorporation of this technique reduces running time of LSACO.

#### 3.3.2 Dynamic Ordering of the Edges

The existing ACO algorithms use a nearest neighbor heuristic to compute the candidate edges to work with. The heuristic rule is built on the assumption that the shorter an edge is, the greater its probability belonging to an optimal tour. This heuristic rule encourages the search process towards short tours, but it reduces the exploration behavior.

Our proposed LSACO algorithm uses a dynamic ordering of candidates to speed up the search process further. In dynamic ordering, when a shorter tour is found, all edges shared by this new tour and the previous shortest tour become the first candidate edges among the list of candidate edges. As a result, the probability of finding the shortest tour is increased. This is done to favor local optima by choosing the recent improved tour's edges.

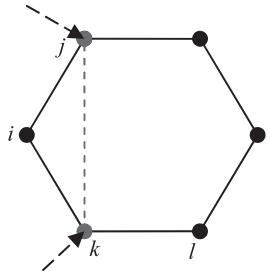


**Fig. 2** Sharing common endpoints.

#### 3.3.3 Other Conditions

The local search will be applied in such a way so that the links of the tours doesn't break. Otherwise, it will take more time to build the links again. So, the new tour that will be obtained by replacing edges must also be a closed tour. Otherwise, opened tour cannot be visited by the ants to find a solution. To maintain the links, a new edge to replace must have a common endpoint with the old edge. That is, edge $(i, j)$ and new edge $(j, k)$ must have a common endpoint and so must $(j, k)$ and $(k, l)$. Thus, this condition helps to maintain the chain of links. Figure 2 illustrates this condition in which the edge $(i, j)$ or $(k, l)$ can be replaced by the edge $(j, k)$.

To reduce computational complexity, several other measures are taken in the local search process. Suppose, if there is more than one edge to select as a new edge, then an untried edge is chosen. If there are more than one alternative for an edge, the one where the cost is minimum, is chosen. This reduces the running time of the local search. To further reduce search time, replaced edges will not be added or added edges will not be deleted.

The search for improvements is stopped if the current tour is the same as a previous solution tour. It saves a lot of running time and it does not affect quality of solutions. If a tour is the same as a previous solution tour, there is no point in attempting to improve it further. The time needed to check that no more improvements are possible is therefore saved. Thus the adaptive local search makes LSACO an efficient algorithm for finding quick optimum solutions for large problem instances.

### 3.4 New Global Updating Rule

The LSACO algorithm uses a new global updating rule to deposit pheromones in the edges of a graph. The aim of deposition is to attract more ants to exploit the visited edges. Our algorithm applies the global updating rule only by a single ant instead of all the ants used in AS [8]. The single ant is either the iteration-best ant or the best-so-far ant, which the algorithm chooses alternatively. The iteration-best ant is chosen as the best ant of the current iteration. The best-so-far ant is selected among best ants from all previous iterations. It is changed at each iteration. The reason for using such an alternation mechanism is to avoid local optima and increase the convergence speed. If the best-so-far ant is allowed to deposit pheromone each time, the search process may fall into local optima. On the other hand, if the iteration-best ant is allowed to deposit pheromone each time, then the number of edges that receive pheromone is larger and the search is less directed. So, LSACO updates pheromone by using the best-so-far ant and the iteration-best ant alternatively. A comparison of using different ants for depositing pheromone is shown in Table 11. The global updating rule of LSACO is given below:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau'_{ij} \tag{6}$$

where $\Delta\tau'_{ij} = 1/C^{bs}$ or $\Delta\tau'_{ij} = 1/C^{ib}$. The variable $C^{bs}$ is the length of the best-so-far tour, while $C^{ib}$ is the length of the iteration-best tour. Since only single ant is allowed to deposit pheromone in each iteration, the computational complexity is only $O(n)$. The LSACO algorithm deposits pheromone on the edges according to Eq. (6), while it decreases pheromone according to Eq. (5). As the algorithm proceeds towards the optimum solution, the shorter paths (edges) will be chosen more by the ants because it will have more pheromone than other edges. In other words, the pheromone levels at the shortest path will converge nearly to one, while they at other edges will converge near to zero. It is important to note that since the best-so-far ant and the iteration-best ant are allowed to deposit pheromone alternatively, LSACO is less prone to fall into local optima.

The computational complexity of the global updating rule of AS [8] was $O(n^2)$. The computational complexity of the global updating rule of other ACO algorithms [9]–[18] are approximately $O(n)$. Here, the computational complexity mainly depends on the number of ants to globally update the pheromone. All the existing ACO algorithms [8]–[18] fall in local optima because of either excessive exploration or exploitation. But LSACO uses the best-so-far ant and the iteration-best ant alternatively to deposit pheromone. So, LSACO is less prone to local optima for solving large combinatorial optimization problems.

## 4. Experimental Studies

This section presents the performance of LSACO on several well-known benchmark problems including symmetric traveling salesman problems [12], [27], asymmetric traveling salesman problems [12], [27] and Hamiltonian cycle problems [28]. To apply LSACO in other combinatorial optimization problems, it is necessary to represent a problem in a graph and then apply all the components of LSACO as described in Sect. 3. These aforementioned traveling salesman problems have been widely used in many previous ACO algorithms. The detailed description of these problems can be obtained from [28], [29]. We here describe experimental results and comparison with other works in this section.

In all the experiments, different parameters of LSACO are set to: $\alpha = 0.1, \beta = 2, q_0 = 0.9$ and $\xi = 0.1$. The number

**Table 1** Experimental results of LSACO on some larger TSP problems [28], [29] over 15 trials.

| Problem (Dimension) | Optimum | Best Length (Error %) | Average Length (Error %) |
|---|---|---|---|
| D2103 (2103) | 80450 | 80450 (0.00%) | 80473 (0.03) |
| Pr2392 (2392) | 378032 | 378032 (0.00%) | 378057 (0.01%) |
| pcb3038 (3038) | 137694 | 137694 (0.00%) | 137712 (0.01%) |
| Fl3795 (3795) | 28772 | 28772 (0.00%) | 28781 (0.03%) |
| fnl4461 (4461) | 182566 | 182566 (0.00%) | 182569 (0.002%) |
| Rl5915 (5915) | 565530 | 565530 (0.00%) | 565693 (0.03%) |
| pla7397 (7397) | 23260728 | 23260728 (0.00%) | 23265210 (0.02%) |
| Rl11849 (11849) | 923288 | 923362 (0.01%) | 923437 (0.02%) |
| D15112 (15112) | 1573084 | 1573282 (0.01%) | 1573362 (0.02%) |

of initial ants $m$ is set randomly between 10 and 20, $m_{\max}$ is set to 1.5 times of $m$ and $\tau$ is set to 10. In tour construction, ants use a candidate list of size 20. However, a candidate list of size 50 is used during adaptive local search for problems with dimension more than 1000. Our algorithm terminates when no further improvement is found after adding an ant. All experiments were run on a machine having Pentium Dual Core processor with 2.2 GHz speed and 1 GB RAM.

### 4.1 Experimental Results

Table 1 shows the results of LSACO over 15 independent trials on different problems. The percentage error in the table is computed by using the following formula:

$$\left(\frac{|best - optimum|}{optimum} \times 100\right)$$

where *optimum* is the length of the shortest possible tour found in [28] and *best* is the best length found by LSACO over 15 trials.

It can be observed from Table 1 that LSACO produces very good results on some large TSP problems. For example, for the pla7397 dataset of the symmetric traveling salesman problem, the best length found by LSACO algorithm was optimal. In fact, the average results of the proposed algorithm was very close to the optimum i.e., average percentage error was 0.01%–0.03%. These results indicate the positive effect of different techniques adopted in our LSACO. The proposed algorithm also shows promising performance on the Hamiltonian cycle problems (Table 2).

### 4.2 Comparison

We compare here the performance of LSACO with some other well known ACO and non-ACO algorithms. These algorithms include ACS-3-opt [12], MMAS-2-opt [13], MMAS-3-opt [13] and cAS [37]. Tables 3–9 and Figs. 3–4 show the comparison of these algorithms. ACS-3-opt, MMAS-3-opt and LSACO were run for different problems of TSPLIB [28], [29] such as fl15177, d2103, pr2392 and pcb3038. The results for each of these problems was then averaged and plotted in Figs. 3 and 4. Problem dimension up to 3038 was taken because above that dimension the time complexity of ACS-3-opt and MMAS-3-opt increased exponentially.

It can be observed from Tables 3–4 that LSACO performed slightly better than ACS-3-opt for symmetric and asymmetric problems. However, LSACO showed significantly better performance in terms of error and time

**Table 2** Performance of LSACO for Hamiltonian cycle problems [28], [29] over 15 trials.

| Name | Dimension | Avg. Time (Seconds) | Success |
|---|---|---|---|
| Alb1000 | 1000 | 0.1 | 15/15 |
| Alb2000 | 2000 | 0.1 | 15/15 |
| Alb3000a | 3000 | 0.1 | 15/15 |
| Alb3000b | 3000 | 0.1 | 15/15 |
| Alb3000c | 3000 | 0.1 | 15/15 |
| Alb3000d | 3000 | 0.1 | 15/15 |
| Alb3000e | 3000 | 0.1 | 15/15 |
| Alb4000 | 4000 | 0.1 | 15/15 |
| Alb5000 | 5000 | 0.2 | 15/15 |

**Table 3** Comparison between LSACO and ACS-3-opt [12] on some symmetric TSP problems [28], [29] over 15 trials.

| Problem (Dimension) | Average Length (% Error ) | | Optimum |
|---|---|---|---|
| | ACS-3-opt | LSACO | |
| D198 (198) | 15782 (0.01%) | 15780 (0.00%) | 15780 |
| Pcb442 (442) | 50812 (0.07%) | 50778 (0.00%) | 50778 |
| Att532 (532) | 27724 (0.13%) | 27686 (0.00%) | 27687 |
| Rat783 (783) | 8998 (2.18%) | 8806 (0.00%) | 8806 |
| fl1577 (1577) | 22741 (2.21%) | 22249 (0.00%) | 22249 |

**Table 4** Comparison between LSACO and ACS-3-opt [12] on some asymmetric TSP problems [28], [29] (over 15 trials).

| Problem (Dimension) | Average Length (%Error) | | Optimum |
|---|---|---|---|
| | ACS-3-opt | LSACO | |
| Ry48p (48) | 14422 (0.00%) | 14422 (0.00%) | 14422 |
| Ft70 (70) | 38679 (0.02%) | 38673 (0.00%) | 38673 |
| Kro124p (124) | 36230 (0.00%) | 36230 (0.00%) | 36230 |
| Ftv170 (170) | 2755 (0.00%) | 2756 (0.04%) | 2755 |

**Table 5** Comparison between LSACO and MMAS-2-opt [13] on symmetric problems of TSP [28], [29] (Over 15 trials).

| Problem (Dimension) | Average Length (%Error) | | Optimum |
|---|---|---|---|
| | MMAS-2-opt | LSACO | |
| Kroa100 (100) | 21284 (0.01%) | 21282 (0.00%) | 21282 |
| D198 (198) | 15791 (0.11%) | 15780 (0.00%) | 15780 |
| Lin318 (318) | 42234 (0.48%) | 42029 (0.00%) | 42029 |
| Pcb442 (442) | 51957 (2.32%) | 50778 (0.00%) | 50778 |
| Att532 (532) | 28503 (2.95%) | 27686 (0.00%) | 27686 |
| Rat783 (783) | 9104 (3.38%) | 8806 (0.00%) | 8806 |

**Table 6** Comparison of LSACO and MMAS-3-opt [13] on asymmetric problems of TSP [28], [29] (Over 15 trials).

| Problem (Dimension) | Average Length (%Error) | | Optimum |
|---|---|---|---|
| | MMAS-3-opt | LSACO | |
| P43 (43) | 5626 (0.09%) | 5621 (0.02%) | 5620 |
| Ry48p (48) | 14501 (0.55%) | 14422 (0.00%) | 14422 |
| Ft70 (70) | 38771 (0.25%) | 38673 (0.00%) | 38673 |
| Kro124p (124) | 36689 (1.27%) | 36230 (0.00%) | 36230 |
| Ftv170 (170) | 2913 (5.73%) | 2755 (0.00%) | 2755 |

**Table 7** Comparison between LSACO and cAS with LK [37] on symmetric problems of TSP [28], [29] (Over 15 trials).

| Problem (Dimension) | Average Length (%Error) | | Optimum |
|---|---|---|---|
| | cAS with LK | LSACO | |
| att532 (532) | 27686 (0.00%) | 27686 (0.00%) | 27686 |
| d1291 (1291) | 50805 (0.008%) | 50801 (0.00%) | 50801 |
| pr2392 (2392) | 378542 (0.13%) | 378057 (0.01%) | 378032 |
| fn14461 (14461) | 184861 (1.25%) | 182569 (0.002%) | 182566 |
| d15112 (15112) | 1596492 (1.48%) | 1573362 (0.02%) | 1573084 |

as the dimension of the problems increases (Figs. 3–4). The proposed LSACO also performed better compared to the MMAS-2-opt, MMAS-3-opt and cAS algorithms (Tables 5–7). The performance of LSACO was far better compared to MMAS-3-opt for problems with a large dimension (Figs. 3–4). Based Because the focus of this paper is on the presentation of the ideas and technical details of LSACO, the detailed comparison with other algorithms using the same heuristics and experimental setup is left as the future work. It is impossible to compare different algorithms fairly unless we reimplement all the algorithms under the same experimental setup and heuristics.

**Table 8** Comparison of base algorithm of LSACO w/o ALS with ACS and MMAS for some small problems of TSPLIB [28], [29] (over 15 trials).

| Problem (Dimension) | Algorithms | Average Length (% Error) | Avg. Time (sec.) |
|---|---|---|---|
| D198 (198) | ACS | 16002 (1.40 %) | 19.8 |
| | MMAS | 16119 (2.15 %) | 23.7 |
| | LSACO w/o ALS | 15780 (0.00%) | 3.9 |
| Pcb442 (442) | ACS | 51623 (1.66 %) | 49.6 |
| | MMAS | 52867 (4.11 %) | 61.4 |
| | LSACO w/o ALS | 50778 (0.00%) | 10.1 |
| Att532 (532) | ACS | 28498 (2.93 %) | 68.2 |
| | MMAS | 29074 (5.01 %) | 88.7 |
| | LSACO w/o ALS | 27688 (0.01%) | 14.5 |
| Rat783 (783) | ACS | 8995 (2.14 %) | 111.2 |
| | MMAS | 9244 (4.97 %) | 147.3 |
| | LSACO w/o ALS | 8815 (0.10%) | 24.8 |
| fl1577 (1577) | ACS | 23147 (4.04 %) | 291.6 |
| | MMAS | 24063 (8.15 %) | 342.6 |
| | LSACO w/o ALS | 22263 (0.06%) | 58.3 |

**Table 9** Comparison of ACS, MMAS and base algorithm of LSACO using ALS (Adaptive Local Search) for some large problems of TSPLIB [28], [29] (over 15 trials).

| Problem (Dimension) | Algorithms | Average Length (% Error) | Avg. Time (sec.) |
|---|---|---|---|
| fl1577 (1577) | ACS+ALS | 22251 (0.01%) | 491.3 |
| | MMAS+ALS | 22257 (0.03%) | 607.9 |
| | LSACO+ALS | 22249 (0.00%) | 97.6 |
| D2103 (2103) | ACS+ALS | 80532 (0.10%) | 1019.5 |
| | MMAS+ALS | 80677 (0.28%) | 1287.7 |
| | LSACO+ALS | 80473 (0.03%) | 173.2 |
| pcb3038 (3038) | ACS+ALS | 138192 (0.36 %) | 1989.1 |
| | MMAS+ALS | 139031 (0.97 %) | 2157.2 |
| | LSACO+ALS | 137712 (0.01%) | 356.5 |
| fnl4461 (4461) | ACS+ALS | 183911 (0.74 %) | 3852.6 |
| | MMAS+ALS | 184639 (1.14 %) | 4639.9 |
| | LSACO+ALS | 182569 (0.002%) | 531.1 |
| Rl5915 (5915) | ACS+ALS | 580127 (2.58 %) | 6813.0 |
| | MMAS+ALS | 587053 (3.80 %) | 7963.8 |
| | LSACO+ALS | 565693 (0.03%) | 762.7 |

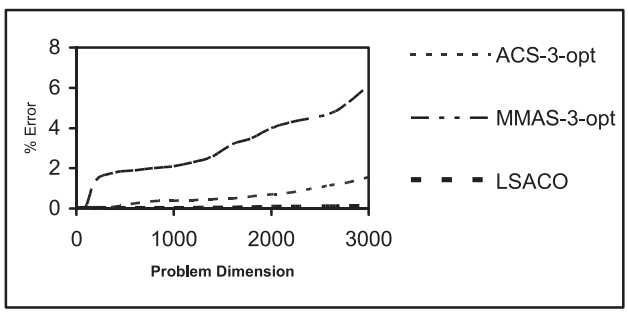


**Fig. 3** Comparison of error rates of LSACO and other best known ACO algorithms with respect to problem dimension.

Table 8 shows the results of LSACO without using the adaptive local search (LSACO w/o ALS) and two other algorithms (i.e., ACS and MMAS) that also do not use any

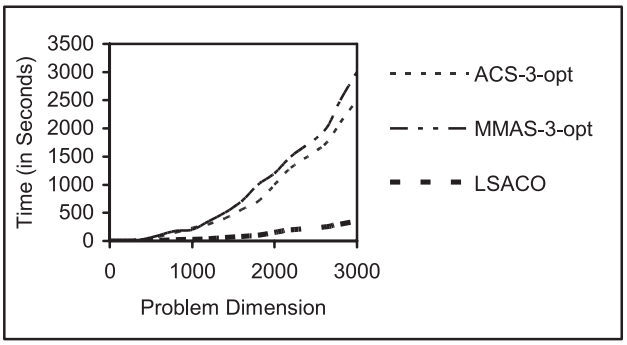


**Fig. 4** Comparison of required time with the best known ACO algorithms to find the optimum solution.

**Table 10** Comparison of LSACO with CLK (Chained Lin-Kernighan) heuristic [35] for some problems of TSPLIB [28], [29] (over 15 trials).

| Problem (Dimension) | Algorithms | Average Length (% Error) | Avg. Time (sec.) |
|---|---|---|---|
| fl1577 (1577) | CLK | 22253 (0.02%) | 170.5 |
| | LSACO | 22249 (0.00%) | 97.6 |
| D2103 (2103) | CLK | 80481(0.04%) | 203.4 |
| | LSACO | 80473 (0.03%) | 173.2 |
| pcb3038 (3038) | CLK | 137694 (0.00%) | 195.7 |
| | LSACO | 137712 (0.01%) | 356.5 |
| fnl4461 (4461) | CLK | 182571 (0.003%) | 360.2 |
| | LSACO | 182569 (0.002%) | 531.1 |
| Rl5915 (5915) | CLK | 565720 (0.03%) | 524.6 |
| | LSACO | 565693 (0.03%) | 762.7 |

local search. It is clear that the base algorithm of LSACO also performs better compared to ACS and MMAS. Table 9 shows the results of LSACO, ACS and MMAS with the same ALS. The performance of LSACO is also better in this case.

So far we have compared LSACO with ACO-based algorithms. It is interesting to know how LSACO performs with respect to non-ACO algorithms. Table 10 shows the results of LSACO and the CLK (Chained Lin-Kernighan) heuristic [35], a non-ACO based algorithm. Our LSACO performs similar to CLK heuristic. For some problems, LSACO performs better while CLK heuristic performs better for other problems. This is reasonable because the CLK heuristic is specially designed for solving TSP problems, while LSACO is designed for any kind of combinatorial optimization problems.

Table 11 shows a comparison of using different ants for depositing pheromone in each global updating phase. There are three cases compared here. In the first case, only the iteration-best ant is allowed to deposit pheromone which made the search less directed. So, it took more iteration to converge. In the second case, only the best-so-far ant was chosen to deposit pheromone which made the search to fall into local optima for some problem instances. In the third case, the iteration-best ant and the best-so-far ant were allowed to deposit pheromone alternatively. As a result, the search was more directed and took less time to converge.

Although the comparison of our LSACO with other algorithms shows novelty of the proposed method, it is not clear the impact of adding ants (an important component of LSACO) on the solution quality of a given problem. To show this effect, we plot the performance of LSACO after adding an ant (Fig. 5). It is clear from this figure that the addition of ants increases the solution quality. However, the addition of ants above a certain level does not improve the solutions quality and this threshold level is usually below 20. When LSACO reaches this threshold level, it stops adding any more ants for solving a given problem.

It can be seen from Fig. 4 that the time requirement of our proposed LSACO is better compared to ACS-3-opt and MMAS-3-opt algorithms. Up to problem size of 500 nodes, the time required by ACS-3-opt [12] and MMAS-3-opt [13] was reasonable. But when the problem size grows more than 500 nodes, the time requirement grows higher whereas our proposed LSACO performed very well even for problem size of more than 2000 nodes. The is because the time complexity of the existing ACO algorithms is nearly $O(n^2)$ whereas our proposed LSACO algorithm's time complexity has been reduced to nearly $O(n)$ from $O(n^2)$. It has been made possible for introduction of several changes inside our

**Table 11** Comparison of using iteration-best ant and best-so-far ant for depositing pheromone in LSACO for some problems of TSPLIB [28], [29] (over 15 trials).

| Problem (Dimension) | Average Length (% Error) | | |
|---|---|---|---|
| | Only iteration-best ant deposits pheromone | Only best-so-far ant deposits pheromone | These two ants deposit pheromone alternatively |
| D2103 (2103) | 80477 (0.03%) | 80481 (0.04%) | 80473 (0.03) |
| pcb3038 (3038) | 137795 (0.07%) | 137822 (0.09%) | 137712 (0.01%) |
| Fl3795 (3795) | 28802 (0.10%) | 28785 (0.05%) | 28781 (0.03%) |
| Rl5915 (5915) | 565691 (0.03%) | 565706 (0.03%) | 565693 (0.03%) |
| pla7397 (7397) | 23265319 (0.02%) | 23265974 (0.02%) | 23265210 (0.02%) |
| D15112 (15112) | 1573491 (0.03%) | 1574239 (0.07%) | 1573362 (0.02%) |

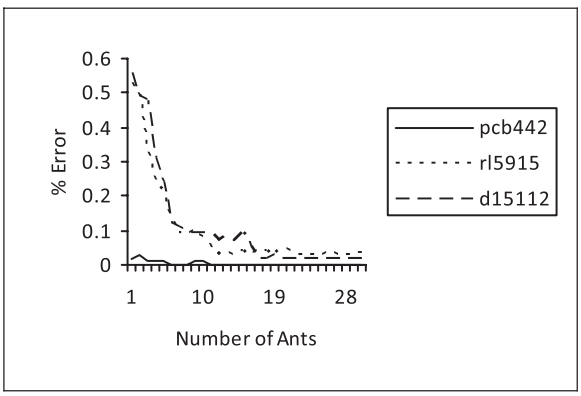


**Fig. 5** Effect of adding ants in LSACO for different problems.

algorithm. LSACO uses candidate list in the tour construction phase for problem size greater than 1000 nodes to reduce the computational complexity of $O(n^2)$ which is very inefficient to solve large problem instances. In existing ACO algorithms, pheromone trails are stored in a matrix with $O(n^2)$ entries. All the entries of this matrix are updated at each iteration. This is a very expensive operation for large problems. So, LSACO updates pheromone only to the edges connecting a node $i$ to nodes belonging to $i$'s candidate list. Hence, the pheromone trails can be updated in $O(n)$ time.

Time complexity of 2-opt and 3-opt local search methods are approximately $O(n^{2.2})$. Since our adaptive local search reduces the number of comparisons and uses different approaches to reduce running time, its time complexity reduces to $O(n)$. So, the average time complexity of our proposed LSACO algorithm is $O(n)$. The time complexity of ACS [12] and MMAS [13] is $O(n^2)$ and the time complexity of ACS-2-opt [12], ACS-3-opt [12], MMAS-2-opt [13] and MMAS-3-opt [13] algorithms are approximately $O(n^{2.2})$.

## 5. Conclusions

The solution quality and the time complexity of ACO algorithms are greatly dependent on their inherent architecture and parameter settings. Although the existing ACO algorithms [8]–[18] show very promising results for solving combinatorial optimization problems, but their performance reduces dramatically when the problem size grows gradually. This paper describes a new algorithm named LSACO for solving large combinatorial optimization problems in a reasonable time and with good quality solutions.

The idea behind LSACO is to put more emphasis on an adaptive local search strategy and to balance the exploration and exploitation of search space in different components of the algorithm by introducing various improvements and modifications. The first improvements include the initialization and updating of pheromone level in new ways in different stages of the algorithm so as to improve the solution quality and to reduce the computational complexity. Besides, the ants are maintained in an adaptive manner unlike the fixed number of ants in the existing ACO algorithms.

The existing local search algorithms are not adaptive. That is, they use fixed value of edge exchange. Since the number of edges is fixed, the solution obtained by local search may not be optimal because it is not possible to know in advance what number of edge exchange to use to achieve the best compromise between running time and quality of solution. Our proposed adaptive local search strategy is better suited due to its ability to cope with different problem sizes and conditions that may arise at different stages during the solution building process of the artificial ants. To make the adaptive local search efficient, the concept of don't look bit and dynamic ordering of the candidates are used.

The proposed LSACO algorithm has a lesser chance to trap into architectural local optima, a common problem suffered by the existing ACO algorithms. All these techniques are adopted in LSACO for designing a robust ACO algorithm with good generalization ability for solving small to large combinatorial optimization problems. Due to these differences with the existing ACO works, LSACO finds good solutions in a reasonable amount of time. Its computational complexity is also significantly lower than the other ACO algorithms. As a result, LSACO can be successfully applied in larger problem instances which were not possible with the existing ACO algorithms.

The extensive experiments reported in this paper have been carried out to evaluate how well LSACO performed on different problems compared to other algorithms. In almost all cases, LSACO outperformed the others (Tables 2–6). In its current implementation, LSACO has a few user-specified parameters although this is not unusual in the field. These parameters, however, are not very sensitive to moderate changes. One of the future improvements to LSACO would be to reduce the number of parameters or make them adaptive.

### References

[1] M. Dorigo and T. Stützle, Ant Colony Optimization, MIT Press, 2004.
[2] M. Zlochin and M. Dorigo, "Model-based search for combinatorial optimization: A comparative study," Parallel Problem Solving from Nature — PPSN VII: 7th International Conference, vol.2439 of Lecture Notes in Computer Science, pp.651–661, Springer-Verlag, Berlin, Germany, 2002.
[3] C. Tsai, C. Tsai, and C. Tseng, "A new approach for solving large traveling salesman problem," Proc. Congress on Evolutionary Computation, vol.2, pp.1636–1641, 2002.
[4] N. Liouane, I. Saad, S. Hammadi, and P. Borne, "Ant systems and local search optimization for flexible job shop scheduling production," International Journal of Computers, Communications & Control, vol.2, no.2, pp.174–184, 2007.
[5] L.M. Gambardella and M. Dorigo, "Ant colony system hybridized with a new local search for the sequential ordering problem," INFORMS Journal on Computing, vol.12, no.3, pp.237–255, 2000.
[6] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," IEEE Trans. Evol. Comput., vol.6, no.4, pp.333–346, Aug. 2002.
[7] C. Blum, "Beam-ACO—hybridizing ant colony optimization with beam search: An application to open shop scheduling," Computer Operations Research, vol.32, pp.1565–1591, 2005.
[8] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," IEEE Trans. Syst. Man Cybern., vol.26, no.1, pp.1–13, 1996.
[9] M. Dorigo, V. Maniezzo, and A. Colorni, "Positive feedback as a search strategy," Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan.
[10] L.M. Gambardella and M. Dorigo, "Ant-Q: A reinforcement learning approach to the traveling salesman problem," Proc. Twelfth International Conference on Machine Learning, pp.252–260, 1995.
[11] M. Dorigo and L.M. Gambardella, "A study of some properties of Ant-Q," Proc. Fourth International Conference on Parallel Problem Solving from Nature, vol.1141 of Lecture Notes in Computer Science, pp.656–665, Berlin, Springer-Verlag, 1996.
[12] M. Dorigo and L.M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," IEEE Trans. Evol. Comput., vol.1, no.1, pp.53–66, April 1997.
[13] T. Stutzle and H. Hoos, "MAX-MIN ant system and local search for combinatorial optimization problems," 2nd International Conference on Metaheuristics, 1997.
[14] B. Bullnheimer, R.F. Hartl, and C. Strauss, "A new rank-based

version of the ant system: A computational study," Central European Journal for Operations Research and Economics, vol.7, no.1, pp.25–38, 1999.
[15] V. Maniezzo, "Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem," INFORMS Journal on Computing, vol.11, no.4, pp.358–369, 1999.
[16] V. Maniezzo and A. Carbonaro, "An ANTS heuristic for the frequency assignment problem," Future Gener. Comput. Syst., vol.16, no.8, pp.927–935, 2000.
[17] C. Blum and M. Dorigo, "The hyper-cube framework for ant colony optimization," IEEE Trans. Syst. Man Cybern. B, Cybern., vol.34, no.2, pp.1161–1172, April 2004.
[18] C. Blum, A. Roli, and M. Dorigo, "HC-ACO: The hyper-cube framework for ant colony optimization," Proc. MIC'2001 — Metaheuristics International Conference, vol.2, pp.399–403, 2001.
[19] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization: Artificial ants as a computational intelligence technique," IEEE Computational Intelligence Magazine, vol.1, no.4, pp.28–39, 2006.
[20] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," Theor. Comput. Sci., vol.344, pp.243–278, 2005.
[21] C. Blum and M. Dorigo, "Search bias in ant colony optimization: On the role of competition-balanced systems," IEEE Trans. Evol. Comput., vol.9, no.2, pp.159–174, April 2005.
[22] M. Dorigo, E. Bonabeau, and G. Theraulaz, "Ant algorithms and stigmergy," Future Gener. Comput. Syst., vol.16, no.8, pp.851–871, 2000.
[23] C. Blum and M. Dorigo, "Deception in ant colony optimization," Proc. ANTS 2004, Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science, vol.3172, pp.119–130, 2004.
[24] C. Blum, "Theoretical and practical aspects of ant colony optimization," Dissertations in Artificial Intelligence, vol.282, 2004.
[25] J. Sun, S. Xiong, and F. Guo, "A new pheromone updating strategy in ant colony optimization," Proc. Third International Conference on Machine Learning and Cybernetics, Shanghai, Aug. 2004.
[26] J.L. Bentley, "Fast algorithms for geometric traveling salesman problems," ORSA Journal of Computing, vol.4, pp.347–411, 1992.
[27] B. Freisleben and P. Merz, "Genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems," Proc. IEEE International Conference on Evolutionary Computation, IEEE Press, pp.616–621, 1996.
[28] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/, date of last visited: 10-Feb-2009.
[29] http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html, date of last visited: 10-Feb-2009.
[30] T. Stutzle and H. Hoos, "Max-min ant system and local search for the traveling salesman problem," Proc. IEEE 4th International Conference on Evolutionary Computation, pp.308–313, 1997.
[31] E.K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," Handbook of Meta-Heuristics, pp.457–474, 2003.
[32] P. Ross, "Hyper-heuristic search methodologies," Introductory Tutorials in Optimization and Decision Support Techniques, Springer, pp.529–556, 2005.
[33] P.C. Chen, G. Kendall, and G.V. Berghe, "An ant based hyper-heuristic for the travelling tournament problem," IEEE Symposium on Computational Intelligence in Scheduling, pp.19–26, April 2007.
[34] D. Ouelhadj and S. Petrovic, "A cooperative distributed Hyper-Heuristic framework for scheduling," IEEE International Conference on Systems, Man and Cybernetics, pp.2560–2565, Oct. 2008.
[35] D. Applegate, W. Cook, and A. Rohe, "Chained Lin-Kernighan for large traveling salesman problems," J. Computing, vol.15, pp.82–92, 2003.
[36] D.S. Johnson and L.A. McGeoch, "The traveling salesman problem: A case study in local optimization," Local Search in Combinatorial Optimization, pp.215–310, 1997.
[37] S. Tsutsui, "cAS: Ant colony optimization with cunning ants," Proc. 9th Conference on Parallel Problem Solving from Nature (PPSN), pp.162–171, 2006.

**Md. Rakib Hassan** received the B.Sc. degree in Computer Science and Engineering from Khulna University of Engineering and Technology (KUET), Khulna, Bangladesh in 2003, and the M.Sc. degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2007. He was a Lecturer from 2003 to 2004 at KUET. He moved to Bangladesh Agricultural University (BAU) as a lecturer of Computer Science and Mathematics department in 2004, where he is now an Assistant Professor. Currently he is a PhD student of Monash University in Australia. His major research interests include swarm intelligence, evolutionary computation, neural networks, distributed systems and cognitive radio networks.

**Md. Monirul Islam** received the B.E. degree from Khulna University of Engineering and Technology (KUET), Khulna, Bangladesh in 1989, the M.E. degree from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 1996, and the Ph.D. degree from the University of Fukui, Japan in 2002. He was a Lecturer and Assistant Professor from 1989 to 2002 at KUET. He moved to BUET as an Assistant Professor of Computer Science and Engineering in 2003, where he is now an Professor. His major research interests include evolutionary robotics, evolutionary computation, neural networks, machine learning, pattern recognition and data mining. He has more than 100 refereed publications. He won the First Prize in The Best Paper Award Competition of the Joint 3rd International Conference on Soft Computing and Intelligent Systems and 7th International Symposium on advanced Intelligent Systems.

**Kazuyuki Murase** is a Professor at the Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, University of Fukui, Fukui, Japan, since 1999. He received ME in Electrical Engineering from Nagoya University in 1978, PhD in Biomedical Engineering from Iowa State University in 1983. He Joined as a Research Associate at Department of Information Science of Toyohashi University of Technology in 1984, as an Associate Professor at the Department of Information Science of Fukui University in 1988, and became the professor in 1992. He is a member of The Japanese Society for Medical and Biological Engineering (JSMBE), The Japan Neuroscience Society (JSN), The International Neural Network Society (INNS), and The Society for Neuroscience (SFN). He serves as a Board of Directors in Japan Neural Network Society (JNNS), a Councilor of Physiological Society of Japan (PSJ) and a Councilor of Japanese Association for the Study of Pain (JASP).