Ant Colony Optimization Based on An Improvement Local Search Strategy

Xiaoyong LIU1,2,3,†

1 National Science Library, Chinese Academy of Sciences, Beijing, 100190,
2 Department of Computer Science, Guangdong Polytechnic Normal University, Guangzhou, Guangdong, 510665,
3Graduate University of Chinese Academy of Sciences, Beijing 100049

Abstract

The traveling salesman problem (TSP) has been an extensively studied problem for a long time and has attracted a considerable amount of research effort. Ant colony system (ACS) algorithm is better method to solve TSP. This paper presents HACS-LS, a hybrid ant colony optimization approach combined with an improvement local search algorithm, applied to traveling salesman problem. The improvement local search algorithm couples heap sort algorithm with 2.5-opt strategy. Four test datasets from TSPLIB, eil76, kroA100, bayg25 and eil51 are chosen to verify the effectiveness of HACS-LS. Experimental results show that HACS-LS performs better than ACS-LS and ACS. The average lengths obtained by HACS-LS are all shorter than those found by ACS-LS and ACS. And then HACS-LS spends less time solving problems than the other algorithms. So the improvement local search algorithm is feasible because of its acceptable time cost.

Keywords: Ant Colony Optimization; Local Search; Traveling salesman Problem; Heap Sort Algorithm

1. Introduction

Ant colonies, and more generally social insect societies, are distributed systems that, in spite of the simplicity of their individuals, present a highly structured social organization. As a result of this organization, ant colonies can accomplish complex tasks that in some cases far exceed the individual capabilities of a single ant. [1-3] In real life, groups of ants are capable of collectively finding shortest paths to food resources via leaving pheromone trails, in the way that iteratively more and more ants follow the shortest paths.

The principle of Ant Colony Optimization (ACO) algorithms is based on the way ants search for food. Each ant takes into consideration (probabilistic choice) pheromone trails left by all other ant colony members which preceded its course, the pheromone trail being a trace, a smell left by every ant on its way. This pheromone evaporates with time, and therefore the probabilistic choice for each ant changes with time. After many ant courses, the path to the food will be characterized by higher pheromone traces and thus all ants will follow the same path. This collective behaviors, based upon a shared memory among all colony ants could be adapted and used for solving combinatorial optimization problems with the following analogies [4].

The real ant search space becomes the space of the combinatorial problem solutions.

The amount of food inside a source becomes the evaluation of the objective function for the corresponding solution.

The pheromone trails become an adaptive shared memory.

† Corresponding author.
Email address: liugucas@gmail.com (Xiaoyong LIU)

1553-9105/ Copyright © 2010 Binary Information Press
October, 2010
As shown in Fig. 1, two ants start from their nest in search of food source at the same time to different directions. One of them chooses the path that turns out to be shorter while the other takes the longer sojourn. The ant moving in the shorter path returns to the nest earlier and the pheromone deposited in this path is obviously more than what is deposited in the longer path. Other ants in the nest thus have high probability of following the shorter route. These ants also deposit their own pheromone on this path. More and more ants are soon attracted to this path and hence the optimal route from the nest to the food source and back is very quickly established. Such a pheromone-meditated cooperative search process leads to the intelligent swarm behavior.[5]

Intuitively, the TSP is the problem of a salesman who, starting from a city, wants to find a shortest tour that takes him through a given set of customer cities and finally returns to the first city, visiting each customer city exactly once. TSP is defined as follows: a salesman visits \( n \) cities (nodes), having distances \( d_{ij} \). Finding the shortest tour among all these closed tours is a NP-complete problem.

The TSP can be represented by a complete weighted graph \( G = (N, A) \) with \( N \) being the set of nodes representing the cities, and \( A \) being the set of arcs. Each arc \( (i, j) \in A \) is assigned a value \( (\text{length})d_{ij} \), which is the distance between cities \( i \) and \( j \), with \( i, j \in N \). Ant colony optimization (ACO) problems could therefore be encoded as finding the shortest path in the graph \( G \). One of the first applications of ACO was the traveling salesman problem (TSP) and its variants, such as PTSP, ATSP, GTSP and etc.[6-13]

2. ACS, ACS-LS and HACS-LS

2.1. ACS

Ant colony system (ACS) algorithm was first proposed by Dorigo in 1997.[7] Informally, ACS works as follows: \( m \) ants are initially positioned on \( n \) cities chosen according to some initialization rule (e.g., randomly). Each ant builds a tour (i.e., a feasible solution to the TSP) by repeatedly applying a stochastic greedy rule (the state transition rule). While constructing its tour, an ant also modifies the amount of pheromone on the visited edges by applying the local updating rule. Once all ants have terminated their tour, the amount of pheromone on edges is modified again (by applying the global updating rule). As was the case in ant system, ants are guided, in building their tours, by both heuristic information (they prefer to choose short edges), and by pheromone information: An edge with a high amount of pheromone is a very desirable choice. The pheromone updating rules are designed so that they tend to give more pheromone to edges which should be visited by ants.

In the general case, ACS algorithm applies the artificial ants concept, it is represented by the following steps:
Step 1. Initialization of parameters.
Step 2. Construction of solutions.
Step 3. Pheromone updating rule.
Step 4. Return to Step 2 until a given stopping criterion satisfied.

If adding local search algorithm after step 2, the new algorithm is named ACS-LS.

2.1.1. State Transition Rule

In ACS the state transition rule is as follows: an ant positioned on node \( i \) chooses the city \( j \) to move to by applying the rule given by function (1).

\[
p_{ij}^k(t) = \begin{cases} 
\frac{[\tau_{ij}(t)]^{\alpha} [\eta_{ik}(t)]^{\beta}}{\sum_{j \in J_k(i)} [\tau_{ij}(t)]^{\alpha} [\eta_{ij}(t)]^{\beta}} & \text{if } j \in J_k(i) \\
0 & \text{otherwise}
\end{cases}
\]  

(1)

Where,
- \( p_{ij}^k(t) \) is the state transition probability with which the ant \( k \) moves from city \( i \) to city \( j \) in \( t \)-th iteration.
- \( \tau_{ij}(t) \) is pheromone value between city \( i \) and city \( j \) in \( t \)-th iteration.
- \( \eta_{ij} = \frac{1}{d_{ij}} \).
- \( J_k(i) \) is the set of cities that ant \( k \) hasn’t been visited when it is at city \( i \).
- \( \alpha \) and \( \beta \) are the parameters and denote heuristic value.

2.1.2. Pheromone Updating Rule

While building a solution (i.e., a tour) of the TSP, ants visit edges and change their pheromone level by applying the updating rule of function (2).

\[
\tau_{ij}(t+1) = \rho \tau_{ij}(t) + \Delta \tau_{ij}(t)
\]

(2)

Where,
- \( \Delta \tau_{ij}(t) = \sum_{k=1}^{N} \Delta \tau_{ij}^k(t) \)
- \( \Delta \tau_{ij}^k(t) = \begin{cases} 
Q & \text{if } t \rightarrow t + 1, \text{city } i \rightarrow \text{city } j \\
\frac{Q}{L_k} & \text{otherwise}
\end{cases} \)

\( N \) is number of ants.
- \( Q \) is a constant and denotes the strength of pheromone.
- \( L_k \) is length of tours that ant \( k \).
- \( \rho \) is residual pheromone value, \( \rho < 1 \).

2.2. Local Search Algorithm

Dorigo and Stutzle has proved that it’s effective that applying local search algorithms in ACS for TSP\(^1\). For the symmetric TSP the neighborhood can be defined by removing a fixed number of \( k \) arcs and replacing them with \( k \) other arcs, leading to a \( k \)-opt algorithm. A straightforward first-improvement algorithm for the TSP searches the neighborhood consisting of all possible \( k \)-opt moves until an improving move is found, replaces the current tour by the new one and continues. If the tour cannot be improved any
more, the whole neighborhood has to be examined to establish local optimality.\textsuperscript{[14]}

The $k$-opt neighborhood consists of those tours that can be obtained from a tour $L$ by replacing at most $k$ of its arcs. Stutzle preferred that an important class of neighborhood structures for combinatorial optimization problems is that of $k$-exchange neighborhoods. The $k$-exchange neighborhood of a candidate solution $L$ is the set of candidate solutions $L_0$ that can be obtained from $L$ by exchanging $k$ solution components. The $k$-exchange neighborhood is the obvious generalization in which a set of $k$ arcs is replaced by a different set of $k$ arcs.\textsuperscript{[1]}

Fig.2 3-exchange Example

Fig.2 gives an example of one specific 3-exchange: the triple arcs (B,C), (C,D) and (F,G) are removed and replaced by the triple arcs (B,D), (F,C) and (C,G).

2.5-opt is a local search algorithm that includes a strongly restricted version of a 3-opt move on top of a 2-opt local search. When checking for an improving 2-opt move, it is also checked whether inserting the city between a city $i$ and its successor, as illustrated in the figure below, results in an improved tour. 2.5-opt leads only to a small, constant overhead in computation time over that required by a 2-opt local search, but it leads to significantly better tours.\textsuperscript{[1]}

2.3. HACS-LS

Usually, ACS-LS spends more times calculating the best tours than ACS, so this paper presents a new local search algorithms that combines with heap sort algorithm. Heap sort algorithm begins by building a heap out of the data set, and then removing the largest item and placing it at the end of the partially sorted array. After removing the largest item, it reconstructs the heap, removes the largest remaining item, and places it in the next open position from the end of the partially sorted array. This is repeated until there are no items left in the heap and the sorted array is full. Elementary implementations require two arrays - one to hold the heap and the other to hold the sorted elements.\textsuperscript{[15]}

HACS-LS can be described as follow,

**Step 1.** Initialization of parameters.

**Step 2.** Construction of solutions.

**Step 3.** Local search algorithm.

Calculating nearest neighbor lists for each city by **Heap sort** in city array Obtain new tour by
using 2.5-opt

**Step 4.** Pheromone updating rule.

**Step 5.** Return to 2 until a given stopping criterion is satisfied.

3. Experimental Results

3.1. Parameters

The algorithm was implemented using Visual C++ 6.0. All experiments have been executed on a PC with 2.0GHz CPU, 1GB DDR RAM.

In three algorithms, six parameters: \( N_{\text{ants}} \), \( \alpha \), \( \beta \), \( \rho \), \( Q \) and \( N_{\text{iteration}} \), is fixed. Table 1 lists the appropriate values of these parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N_{\text{ants}} )</td>
<td>50</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>1</td>
</tr>
<tr>
<td>( \beta )</td>
<td>5</td>
</tr>
<tr>
<td>( \rho )</td>
<td>0.9</td>
</tr>
<tr>
<td>( Q )</td>
<td>100</td>
</tr>
<tr>
<td>( N_{\text{iteration}} )</td>
<td>200</td>
</tr>
</tbody>
</table>

3.2. Results

Three algorithms, HACS-LS, ACS-LS and ACS, are run ten times respectively. Fig.3, Fig.4, Fig.5 and Fig.6 show that best tours are obtained in a numerical experiment by those algorithms. It is obvious that HACS-LS can arrive at the shortest tours in four standard datasets, eil 76, kroA 100, brag 26 and eil 51. In eil 76, HACS-LS spends less time getting the shortest tours. Comparing to ACS-LS and HACS-LS, ACS acquires the longest tours. So, the performance of HACS-LS is best among three algorithms in numerical experiments.

![Fig.3 Length in Best Tours of eil76 by Three Algorithms](image-url)
Table 2 Eil76

<table>
<thead>
<tr>
<th></th>
<th>Best</th>
<th>Worst</th>
<th>Average</th>
<th>Std</th>
<th>Average of running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>HACS-LS</td>
<td>548.7004</td>
<td>561.2204</td>
<td>553.5099</td>
<td>3.6514</td>
<td>13.7684</td>
</tr>
<tr>
<td>ACS-LS</td>
<td>552.7222</td>
<td>573.6948</td>
<td>568.1634</td>
<td>6.4239</td>
<td>14.7622</td>
</tr>
<tr>
<td>ACS</td>
<td>563.3341</td>
<td>575.904</td>
<td>571.9029</td>
<td>4.2460</td>
<td>13.4776</td>
</tr>
</tbody>
</table>
Table 3 KroA100

<table>
<thead>
<tr>
<th></th>
<th>Best</th>
<th>Worst</th>
<th>Average</th>
<th>Std</th>
<th>Average of running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>HACS-LS</td>
<td>21450.4116</td>
<td>21967.8186</td>
<td>21722.7291</td>
<td>139.6021</td>
<td>23.5185</td>
</tr>
<tr>
<td>ACS-LS</td>
<td>21716.6859</td>
<td>22412.9999</td>
<td>22029.2284</td>
<td>222.7305</td>
<td>24.8668</td>
</tr>
<tr>
<td>ACS</td>
<td>22525.4246</td>
<td>23225.1811</td>
<td>22871.7507</td>
<td>285.8522</td>
<td>23.1932</td>
</tr>
</tbody>
</table>

Table 4 Bayg29

<table>
<thead>
<tr>
<th></th>
<th>Best</th>
<th>Worst</th>
<th>Average</th>
<th>Std</th>
<th>Average of running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>HACS-LS</td>
<td>9077.9177</td>
<td>9212.2267</td>
<td>9144.4472</td>
<td>43.7277</td>
<td>2.3574</td>
</tr>
<tr>
<td>ACS-LS</td>
<td>9120.3388</td>
<td>9234.2089</td>
<td>9156.4824</td>
<td>32.8039</td>
<td>2.7136</td>
</tr>
<tr>
<td>ACS</td>
<td>9234.2089</td>
<td>9318.6158</td>
<td>9275.3318</td>
<td>23.8052</td>
<td>2.1682</td>
</tr>
</tbody>
</table>

Table 5 Eil51

<table>
<thead>
<tr>
<th></th>
<th>Best</th>
<th>Worst</th>
<th>Average</th>
<th>Std</th>
<th>Average of running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>HACS-LS</td>
<td>429.1179</td>
<td>432.8186</td>
<td>432.0722</td>
<td>2.9629</td>
<td>7.1512</td>
</tr>
<tr>
<td>ACS-LS</td>
<td>429.7371</td>
<td>451.5417</td>
<td>437.8952</td>
<td>5.4391</td>
<td>7.9479</td>
</tr>
<tr>
<td>ACS</td>
<td>447.0601</td>
<td>462.2207</td>
<td>453.8439</td>
<td>5.0557</td>
<td>6.8216</td>
</tr>
</tbody>
</table>

4. Conclusion

In this paper I have proposed a robust ACO algorithm for traveling salesman problem. The algorithm
is based on ant colonies optimization and 2.5-opt local search strategy. Heap sort algorithm is applied
in the local search strategy. Four standard TSP dataset, eil 76, kroA 100, bayg 29 and eil 51, are
chosen to test the performance of the new algorithm, HACS-LS. Numerical experiments show that
HACS-LS has more stable than ACS-LS and ACS and get better solutions for traveling salesman
problem. Therefore, HACS-LS is an effective algorithm for TSP.

The future work will focus on studying different value of parameters to affect performance of
HACS-LS.

Acknowledgement

The author would like to thank anonymous reviewers for their constructive and enlightening comments,
which improved the manuscript. This work has been supported by grants from Program for Excellent and
Creative Young Talents in Universities of Guangdong Province (LYM08074 and LYM09097).

Reference

73–81,1997.


