Solving Geometric TSP with Ants

Thang N. Bui and Mufit Colpan
Computer Science Program
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
{tbui, muc135}@psu.edu

ABSTRACT

This paper presents an ant-based approach for solving the Traveling Salesman Problem (TSP). Novel concepts of this algorithm that distinguish it from the other heuristics are the inclusion of a preprocessing stage and the use of a modified version of an ant-based approach with local optimization in multi stages. Experimental results show that this algorithm outperforms ACS [1] and is comparable to MMAS [4] for Euclidean TSP instances. Of the 40 instances of Euclidean TSP from TSPLIB [5] that were tested, this algorithm found the optimal solution for 37 instances. For the remaining instances, this algorithm returned solutions that were within 0.3% of the optimum.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Design, Algorithms

Keywords

Traveling Salesman Problem, Ant System

1. INTRODUCTION

The Traveling Salesman Problem (TSP) is the prototypical optimization problem that is difficult to overcome but is both short and easy to state: given n cities such that there is a direct edge between any two of them, and each edge is assigned a cost, the task is to find the cheapest way of visiting each city exactly once starting and finishing at the same city.

The set of edges used in traveling the cities as stated is referred to as a *Hamiltonian tour* and the sum of all the edge costs in the tour is called the *tour cost*.

The TSP asks for the smallest cost Hamiltonian tour in a complete undirected graph with positive costs on the edges. More formally, the TSP is defined as follows:

Input: A complete undirected graph $G = (V, E)$ with vertex set V, edge set E, and a weight function $w : E \rightarrow \mathbb{R}^+$.

GECCO'05, June 25–29, 2005, Washington, DC, USA.
ACM 1-59593-010-8/05/0006.

Output: A minimum weight Hamiltonian tour of G.

There are many different variations of the traveling salesman problem. In this paper we consider a version of TSP called *geometric* TSP, in which vertices of the graph are points in the Euclidean plane and the weight of an edge is just the Euclidean distance between the two end points of the edge. We present an algorithm that is a combination of an ant system and an efficient local optimization algorithm. Additionally, a preprocessing stage that allows us to deal with larger input instances efficiently was added. The performance of the algorithm on a set of standard benchmarks for the geometric TSP showed that the algorithm is quite competitive against existing heuristics. In fact, it found 37 optimal solutions out of the 40 instances that were tested.

2. THE ALGORITHM

There are two main ideas in our algorithm: the inclusion of a preprocessing stage and the use of a modified ACS with local optimization in multi stages. The input graph is first partitioned into subgraphs of size 3, i.e., triangles. A new graph is created based on the centroids of the triangles. An ant colony system algorithm, called mACS and described later in this section, is used to create a tour of this graph. The tour is then converted into an initial tour for the original graph. A local optimization algorithm, such as 2-opt, 3-opt or Lin-Kernighan, is then used to improve this original tour. Finally, we run the mACS algorithm on the original graph using the tour just constructed as a starting tour.

The algorithm consists of two main phases. In the first phase a good starting tour is created. This tour is then used as the starting point for the second phase to create a better tour.

The first phase consists of four stages. The first stage compacts the given input graph into a smaller graph. The objective is to reduce the problem size so that we can find an initial tour, quickly enabling us to deal with larger input graph. In the second stage, the compacted graph is passed into mACS to find a TSP tour. The tour found by mACS is a TSP tour of the compacted graph. This tour is then converted into a tour for the original input graph in the third stage, and is locally optimized in the fourth stage. The resulting TSP tour is now a reasonably good tour of the input graph and is used as the starting point for phase two.

In phase two, the mACS algorithm is run using the original input graph and the TSP tour from phase one. The tour found by mACS in this phase is returned as the solution to the input instance.

The main idea of using phase one is based on the expectation that a good starting tour for mACS will allow mACS to find a better solution and converge in much less time. The overall effect is that we can deal with larger input instances, producing good solutions in small amount of times. The ACS-TSP algorithm is given in Figure 1.

ACS-TSP($G = (V, E, w)$)
Phase 1.
$(G' = (V', E', w')) \leftarrow$ Compact(G)
$tour \leftarrow$mACS $(G', \emptyset)$
$tour \leftarrow$ConvertTour$(G, G', tour)$
$tour \leftarrow$LocallyOptimize$(tour)$
Phase 2.
$bestTour \leftarrow$mACS $(G, tour)$
return $bestTour$

Figure 1: The ACS-TSP Algorithm

---

mACS is a modified version of the ACS [1]. Our modifications include flexible state transition and pheromone updating rules. We also introduce explicit mechanisms for escaping from local optima as well as increasing time efficiency.

In ACS, the parameters q_0, α, and ρ are constants that do not change during the execution of the ACS algorithm. Intuitively, q_0 determines the relative importance of exploitation versus biased exploration. Whereas, α and ρ determine the desirability of the edges.

In mACS, ants explore more at the beginning and toward the end of the algorithm they tend to exploit more, utilizing the information that have been accumulated. We accomplish this by allowing q_0, α and ρ to change in each cycle.

As q_0 gets larger exploitation occurs more. More exploitation may direct ants to use the edges of the global best tour more frequently. To avoid a local optimum, we perturb the global best tour by erasing some memory from some percentage of randomly chosen edges of the tour. Additionally, when ants find the same tour over and over again or do not improve the global best tour after a certain number of iterations we encourage the exploration of edges not used frequently.

3. EXPERIMENTAL RESULTS

We report the results obtained by running ACS-TSP on a collection of benchmark problems from TSPLIB [5], a database of benchmark instances for the TSP problems, and compare them against the known optimal solutions and best known results from two other algorithms: ACS and MMAS [4].

Our algorithm was implemented in C++ and run on a Pentium IV 3.2GHz processor PC with 1GB of RAM. We tested our algorithm on symmetric Euclidean TSP instances. All tests have been carried out for 2500 cycles with 20 trials per instance. The value of the parameter β was 2 and the number of ants was 10. We have purposely chosen the values for β and the number of ants as specified in order to obtain an equal basis for comparison with ACS.

The results shown in Table 1 list the best tour length, BEST, and the average tour length, AVERAGE, found by our algorithm for 20 trials.

Table 1: ACS-TSP solution quality

		ACS-TSP			
INSTANCE	OPTIMUM	BEST	AVERAGE	*ACS*	*MMAS*
att48	10,628.00	10,628.00	10,628.00		
att532	27,686.00	27,686.00	27,701.45	28,147.00	27,686.00
a280	2,579.00	2,579.00	2,579.10		
berlin52	7,542.00	7,542.00	7,542.00		
bier127	118,282.00	118,282.00	118,282.00		
ch130	6,110.00	6,110.00	6,122.60		
ch150	6,528.00	6,528.00	6,538.05		
d198	15,780.00	15,780.00	15,780.65	15,888.00	15,780.00
d1291	50,801.00	50,801.00	50,938.65		50,801.00
eil51	426.00	426.00	426.60	426.00	426.00
eil76	538.00	538.00	538.00		
eil101	629.00	629.00	629.40		
fl1400	20,127.00	20,127.00	20,214.35		
fl1577	22,249.00	22,294.00	22,551.00	22,977.00	22,286.00
kroA100	21,282.00	21,282.00	21,282.00	21,282.00	21,282.00
kroA200	29,368.00	29,368.00	29,392.25		
kroB150	26,130.00	26,130.00	26,139.15		
kroB200	29,437.00	29,437.00	29,509.25		
kroC100	20,749.00	20,749.00	20,749.00		
kroD100	21,294.00	21,294.00	21,294.00		
kroE100	22,068.00	22,068.00	22,088.45		
lin105	14,379.00	14,379.00	14,379.00		
lin318	42,029.00	42,029.00	42,161.45		42,029.00
oliver30	420.00	420.00	420.00		
pcb442	50,778.00	50,778.00	50,823.15		50,778.00
pcb1173	56,892.00	56,896.00	57,022.05		56,896.00
pr76	108,159.00	108,159.00	108,159.00		
pr107	44,303.00	44,303.00	44,347.70		
pr124	59,030.00	59,030.00	59,032.30		
pr136	96,772.00	96,772.00	96,787.45		
pr144	58,537.00	58,537.00	58,537.00		
pr152	73,682.00	73,682.00	73,729.60		
pr226	80,369.00	80,369.00	80,369.00		
pr439	107,217.00	107,217.00	107,297.85		
rat575	6,773.00	6,773.00	6,781.75		
rat783	8,806.00	8,806.00	8,822.80		8,806.00
tsp225	3,916.00	3,916.00	3,917.40		
u159	42,080.00	42,080.00	42,080.00		
u1060	224,094.00	224,484.00	225,430.75		224,455.00
vm1084	239,297.00	239,297.00	239,383.50		

4. CONCLUSION

Experimental results show that our hybrid ant system algorithm ACS-TSP outperforms ACS in solution quality and is comparable to MMAS for symmetric Euclidean TSP instances. The running time of our algorithm can be improved further by using more sophisticated data structures such as those suggested by [2, 3].

5. REFERENCES

[1] Dorigo, M. and L. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," IEEE Transactions on Evolutionary Computation, 1(1), 1997, pp. 53–66.

[2] Bentley, J.L., "Fast Algorithms for Geometric Traveling Salesman Problems," ORSA Journal on Computing, 4(4), 1992, pp. 387–411.

[3] Fredman, M. L., D. S. Johnson, L. A. McGeoch, and G. Ostheimer, "Data Structures for Traveling Salesmen," Journal of Algorithms, 18, 1995, pp. 432–479.

[4] Stutzle, T. and H. Hoos, "Improvements on the Ant-System : Introducing the MAX-MIN Ant System," Proceedings of Artificial Neural Nets and Genetic Algorithms, Springer Verlag, Wien, Austria, 1998, pp. 245–249.

[5] *TSPLIB: Library of Sample Instances for the TSP.* University of Heilderberg, Department of Computer Science, February 2005, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.