2011 International Conference on Advancements in Information Technology
With workshop of ICBMG 2011
IPCSIT vol.20 (2011) © (2011) IACSIT Press, Singapore

A Recursive Ant Colony System Algorithm for the TSP

Nitish Sabharwal [1+], Harshit Sharma [2]

[1] Dept. of Computer Science Birla Institute of Technology and Science, Pilani, India
[2] Dept. of Information Technology Ansal Institute of Technology, Gurgaon, India

Abstract. In this paper we have studied the application of recursive techniques for ant colony optimization algorithms and local heuristics to find solutions for the traveling salesman problem. We propose a Recursive Ant Colony System (RACS) algorithm to improve the performance of Ant Colony System algorithm for large scale Traveling Salesman problems by partitioning the feasible tours obtained and applying the algorithm recursively on smaller paths to find good solutions for the problems by improving the exploration efficiency of the ants. Computer simulations demonstrate that the RACS algorithm is capable of generating good solutions to both symmetric and asymmetric instances of the TSP. The results obtained by the proposed algorithm have also been compared with the normal Ant Colony System (ACS) algorithm for finding solutions to the symmetric Traveling Salesman Problem.

Keywords: Travelling Salesman Problem, meta-heuristic optimization, Swarm Intelligence, Ant Colony System, Partitioning.

1. Introduction

The Travelling Salesman Problem (TSP) is an NP-hard problem and is one of the most famous and well-studied problem in the combinatorial optimization field. It has several applications even in its purest formulation, such as planning logistics, threading of scan cells in a testable VLSI circuit [1], X-ray crystallography [2], etc. In standard TSP, the goal is to find the minimum length Hamiltonian cycle through a set of n cities, given the distances between all pairs of cities. No exact polynomial time algorithms exist for solving the TSP, however many approximate algorithms applying various heuristic approaches have been proposed in the past. Examples of meta-heuristics algorithms used for TSP include simulated annealing [3], tabu search [4], iterated local search [5], evolutionary computation [6], and ant colony optimization(ACO). ACO, proposed by Dorigo *et al*. [7], is a recent member of the family of meta-heuristic algorithms and is widely used to solve combinatorial optimization problems. ACO is inspired by behaviour of real ants. The basic idea is to imitate the stigmergic behavior which enables ants to lay down pheromones so as to exchange information about environment and solve the problem using collective intelligence [8].

In this paper we use Ant Colony optimization for finding solutions to the TSP and propose an algorithm, the Recursive Ant Colony System (RACS), which applies the Ant Colony System (ACS) algorithm recursively on smaller cost matrices obtained by partitioning initial larger cost matrices of the problem. The new algorithm can effectively be used to tackle large scale optimization problems which can get stagnated on a poor solution when solved using the basic Ant Colony algorithms. The paper has been further organized as follows: in section II a brief overview of ant colony system is provided followed by the description of the RACS algorithm in section III. Section IV reports on the experimental results and the comparisons with other algorithms followed by section V which lists the conclusions.

[+] Corresponding author.Tel.: + 918007552346;
E-mail address: nitishsab@gmail.com.

2. Background

2.1. Ant System

Ant System is the forebear of the Ant colony System optimization algorithms which was first applied to Travelling Salesman Problem [9]. Ant Colony System (ACS) builds on the Ant System, which is inspired from foraging behaviour of ant species. Real ants are capable of finding shortest path to food source through stigmergic communication and without following visual cues [10]. Foraging ants deposit chemicals (pheromones) while walking in search of food source, thus increasing the probability of other ants following the same path. This communication is a pheromone mediated indirect communication, and the ants exploit the pheromone information to help them to find a shortest path in the search of food.

2.2. TSP

A general Travelling Salesman Problem can be represented by a complete weighted directed graph G = (V, E), where V = {1, 2, 3....n} is a set of vertices representing cities and E being the set of arcs fully connecting the nodes [11]. In addition, E is associated with the set C representing costs L, where L_{ij} is the metric of the distance between the cities i and j. The objective of the TSP is to find shortest closed tour in V such that each city is visited once and only once.

2.3. Ant Colony System

The major difference between ACS and the AS is in the form of a pseudo-random proportional rule mentioned in (1). ACS when applied to solve TSP work as follows: Artificial ants are positioned on certain number of cities according to some initialization rule (i.e. random in this case). The structure of general ACS algorithm is shown in following major steps [12] (Table 1):

Table 1. General ACS algorithm

1. Initialize pheromone trails and place *M* ants on the nodes of AS graph
2. Repeat until system convergence
2.1 For *i* = 1 to *n*
2.1.1 For *j* = 1 to *M*
2.1.1.1 Choose the node *s* to move to, according to the transition probability specified in (2).
2.1.1.2 Move the ant-*k* to the node *s*
2.2 Update the pheromone using the pheromone update formula (3)

Let x be an ant assigned a task to take a cyclic tour of all the cities and let $J_k(r)$ be the list of cities associated to x containing cities yet to be visited, where r is the current city. An ant x situated in city r moves to a city s according to the state transition rule as follows:

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \{\tau(r,u).\eta(r,u)^{\beta}\}, & \text{if } q \le q_o \quad \text{(exploitation)} \\ S, & \text{otherwise} \quad \text{(biased exploitation)} \end{cases} \tag{1}$$

where, $\tau(r,u)$ stands for pheromone on the edge (r,u), $\eta(r,u) = 1/\delta(r,u)$ is the desirability of edge(r,u), β is a parameter which determines the relative importance of pheromone versus distance, q is a value chosen randomly with uniform probability in [0,1], and q_0 ($0<=q_0<=1$) is a parameter that decides the probability to make random choices or to exploit the edges with higher pheromones, and S is a random variable selected according to the random proportional rule given below:

$$P_k(r,s) = \begin{cases} \dfrac{\tau(r,u).\eta(r,u)^{\beta}}{\sum_{u \in j_k(r)} \tau(r,u).\eta(r,u)^{\beta}}, & \text{if } s \in J_k(r) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Equations (1) and (2) explain the exploitation of accumulated knowledge related to problem or exploration of new edges during transition. State transition rule helps to favour transitions with higher amount of pheromone trail as well as short edges. While ants visit edges in order to find a solution of the TSP, during their visit to each edge, ants change pheromone level of edges by applying local updating rule as described in equation (3).

$$\tau(r,s) \leftarrow (1-\rho) . \tau(r,s) + \rho . \tau_0 \tag{3}$$

where, $0<\rho<1$ is the coefficient representing pheromone evaporation, and n is the number of cities and $\tau_0 = (n*L_{nn})^{-1}$, where L_{nn} is the tour length produced by nearest neighbour heuristic [13].

After all ants complete their cyclic tour, only the globally best ant (i.e. ant belonging to shortest tour) changes trail following global updating rule as given in equation (4).

$$\tau(r,s) \leftarrow (1- \alpha) . \tau(r,s) + \alpha . \Delta \tau(r,s) \quad (4)$$

where, $0<\alpha<1$ is pheromone decay parameter, L_{gb} is the length of globally best tour, and

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1} & \text{, if } (r,s) \in \text{global best tour} \\ 0 & \text{, otherwise} \end{cases} \quad (5)$$

Global updating rule is similar to a reinforcement learning process as in this case better solutions get higher reinforcement, thus providing high amount of trail to shorter tours.

3. The RACS Algorithm

The Recursive Ant Colony System (RACS) algorithm applies a partitioning scheme to the problem in a manner analogous to the recursive merge sort based on the divide and conquer technique. A 2-dimensional partitioning scheme was proposed by Karp in 1977 which was similar to the construction of a k-d tree data structure [14]. It involved geometrically partitioning the cities and then used local optimization heuristics such as 2-opt and 3-opt for improvement. Alternative partitioning schemes were introduced by Reinelt [11] and Rohe [15], though experimental results concluded that neither of these approaches are preferable to Karp's scheme. We used a tour-based partitioning technique to avoid isolation of sub problems that can occur in geometric partitioning, as discussed in [16].

The algorithm is based on the fact that the efficiency of Ant Colony applications is better for problems of smaller size having less number of cities. This occurs due to the random nature of the algorithm, in which a large number of good random decisions made on weighed choices are required to come together to construct an efficient solution and as the size of the problem increases, so do the number of decisions to be made to generate a single tour. The RACS algorithm partitions the set of all nodes for a problem; say S, into two disjoint sets, say S_1 and S_2, and then proceeds to find solutions independently for the two sub-problems now created by focusing on reducing the lengths of the segments formed by these sets in the original tour, keeping the end points of any new path same as that in the original path. As the search space for these sub-problems gets reduced, resulting from the division of the nodes for the original problem, the exploration efficiency and hence the accuracy of the ACS algorithm is much greater for these sub-problems. The accuracy of the overall solution obtained by the conjunction of the solutions obtained from these sub-problems is upper-bounded by the accuracy of the division of the nodes for each subset, which in turn depends upon the accuracy of the initial candidate tour generated. Thus, the RACS algorithm employs a strategy of generating a candidate tour initially using an iterative ACS procedure, followed by partitioning of the tour and recursive implementation of the ACS and Greedy 2-opt(for symmetric TSPs) algorithms on the sub-problems created at each recursive level, to further improve the candidate solution initially generated.

The recursive implementation can speed up the convergence for a large problem before its stagnation by focussing upon a targeted set of vertices separately and finding convergent paths for these smaller sub-problems for which convergence can be obtained rapidly. Thus, the RACS algorithm is advantageous for larger problems where a convergent path is not easily found in limited time by using solely the Ant Colony algorithms. It can also avoid stagnation behaviour by breaking down the problem and exploring alternate routes for each sub-problem. In this paper we have applied the RACS algorithm on the solutions obtained by applying the ACS algorithm before it reaches stagnation. For the initialization purpose, we have incorporated a nearest neighbour strategy into the ACS in the initial stages of the algorithm. We have also used a Greedy 2-opt edge exchange [11] heuristic for local optimization, implemented as discussed by Nilsson [17], for symmetric TSPs as its time complexity is $O(n^2)$ and it does not affect the overall time complexity of the algorithm.

4. The Proposed Algorithm:

- **Initialization of parameters**

Set the initial pheromone values for all edges as $1(\tau_{ij} = 1 \ \forall \ (i, j) \in J_k)$. Initialize all other parameters, including the number of cycles T that the modified ACS will run for on the top recursive level. Populate the cost matrix with the distances between the cities. Set the Recursion counter RC as 0.

- **Getting an initial optimal tour**

Apply nearest neighbor heuristic to the problem to get an initial optimal value and then apply a modified ACS algorithm on the problem for the next e iterations in which during the construction of a solution an ant would select a city j from city i with probability P_{ij} defined by the following equation,

$$\mathcal{P}_{ij} = \begin{cases} \arg\max\{[\tau(i,j)] * [\eta(i,j)]^{\beta}\}, \text{ if } q<q_0, & \text{(exploitation)} \\ \arg\max\{\eta(i,j)\} \text{ for } j \in J_k \text{ , if } q > q_0 \text{ and } q < q_1, & \text{(nearest neighbor)} \\ S \text{ , otherwise} & \text{(biased exploration)} \end{cases}$$

- **Run normal Ant Colony**

Run the normal Ant Colony System as introduced in section II for $\gamma * T$ iterations, where $0<\gamma<1$, on the input matrix. Store the best tour obtained and its length, say L.

- **Partitioning**

Partition the best tour obtained into two paths, choosing a random node for separation, such that the difference in the number of nodes on each path is less than or equal to 1. Copy the cost matrix and the pheromones of the appropriate edges into the two separate matrices representing the nodes of the partitioned paths. Store the individual values of the two paths in the best tour obtained till now, say L_1 and L_2.

- **Recursive implementation**

Increment RC. Apply the following steps recursively on each path till RC < c, where c specifies the number of levels of recursion the algorithm must run.

a) Apply the ACS method for $\varepsilon * \gamma * T$, iterations, where $0<\varepsilon<1$, on the partitioned path, with a slight modification such that the endpoints of the shuffled paths remain the same as that of the initial segment and there is no edge between them, to improve its length. Store the best tour obtained and its length.
b) Partition the path in a similar way as in step 4 into two separate paths and repeat step 5 for the two partitioned paths.
c) Copy back the updated pheromones into the respective edges for the nodes of the original path and update the best path. Again apply the ACS algorithm on the path for $(1 - \varepsilon)*\gamma*T$ iterations. Store the best solution obtained.
d) Apply Greedy 2-opt edge exchange on the tour obtained previously and store the tour.

- **Updating solution**

Update the best tour by reconnecting the two paths with the same edges that joined them before. Update the pheromones of the original edges with the updated pheromone levels from step 5. Run the conventional ACS algorithm for the remaining $(1- \gamma)*T$ iterations, taking the initial globally best value as the length obtained after joining the two paths. Apply Greedy 2-opt on the final solution obtained.

5. Experimental Results

All problem instances studied as test cases in this paper are TSPs taken from TSPLIB [18] and all algorithms have been implemented using C language on a Linux based platform. For each test case of symmetric TSPs, 10 trials were conducted and for the asymmetric TSPs we conducted 5 trials for each test case. The number of total iterations carried out in a trial was varied considering the size of the problem. The performance of the algorithm depends on the correct tuning of several parameters, namely: α, β, relative importance of trail and attractiveness, ρ, trail persistence, $\tau_{ij}(0)$, initial trail level, and Q, used for defining to be of high quality solutions with low cost. Most of the parameters used in the algorithms are inspired from the literature since they have been found effective [19]. We set the parameters for the ACS as: $\alpha = 1$, $\beta = 6$ for TSP test cases and $\beta = 8$ for the ATSP cases, $\rho = 0.75$, $\tau_{ij}(0) = 1.00$ and $q_0 = 0.85$ for these experiments, except for the initialization phase where we varied q_1 from 0.95 to 0.88 and kept q_0 as 0. For the RACS algorithm, we set the other parameters as, c=2, $\varepsilon = 0.70$, $\gamma = 0.60$.

Below provided tables and graphs summarize the results of the RACS algorithm on a few test cases and compares it with the results obtained by running the ACS component of the algorithm by itself. In Tables 2,

3 and 4, we report the results obtained for TSP and ATSP problems. In the first column we report the problem name, the number of nodes (in parentheses) and the number of evaluations at the top recursive level for RACS in Tables 2 and 4, and the total number of iterations for ACS in Table 3. In the second column we report the best result obtained by the respective algorithms. In the third column we report average result and in the fourth column we report the optimal result. In the last column we give the error percentage, a measure of the quality of the best result found by ACS:

Error % = 100*((Best Result obtained – Optimal result)/Optimal Result).

Table 2. RACS performance for TSP cases.

Problem(nodes) [#evaluations]	RACS Best result	RACS average	Best known result	% Error
Pr107(107) [150000]	44385	44484.52	44303	0.19
Ch150(150) [200000]	6537	6574.35	6528	0.14
Pr439(439) [800000]	108145	109336.53	107217	0.86
Rat783(783) [800000]	8976	9035.14	8806	1.93

Table 3. ACS performance for TSP cases.

Problem(nodes) [#evaluations]	ACS Best result	ACS average	Best known result	% Error
Pr107(107) [300000]	44405	44535.65	44303	0.23
Ch150(150) [400000]	6547	6593.24	6528	0.29
Pr439(439) [1000000]	108865	109839.21	107217	1.54
Rat783(783) [1000000]	9024	9104.36	8806	2.52

From the tables 2 and 3, we can see that the improvement of the tour by the RACS increases as the size of the problem increases. Figure 1 is a graph depicting the reduction of best tour length after the initialization phase by the ACS algorithm on a test case instance of pr439 taken from TSPLIB. Figures 2 and 3 depict the value of the best path lengths formed from the breakdown of the best tour of the same test case instance by applying ACS on the partitioned segments. From the graphs, it can be observed that the reduction of the optimal tour length gets accelerated once the best tour about to get stagnated is decomposed into two segments which are then improved upon by the ACS algorithm.

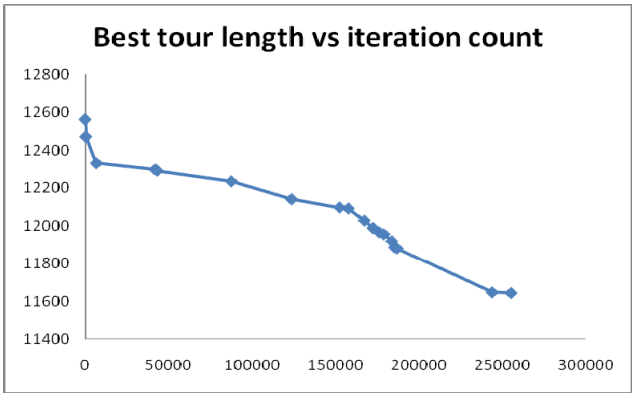

Fig. 1: Behavior of the ACS algorithm for a test case of the pr439 problem at the top level of the algorithm.

Table 4. RACS performance for ATSP cases

Problem(nodes) [#evaluations]	RACS Best result	RACS average	Best known result	% Error
Ft53(53) [650000]	7027	7104.38	6905	1.77
Kro124(100) [1050000]	37085	37792.12	36230	2.36
Ftv170(171) [1500000]	2836	2916.53	2755	2.94

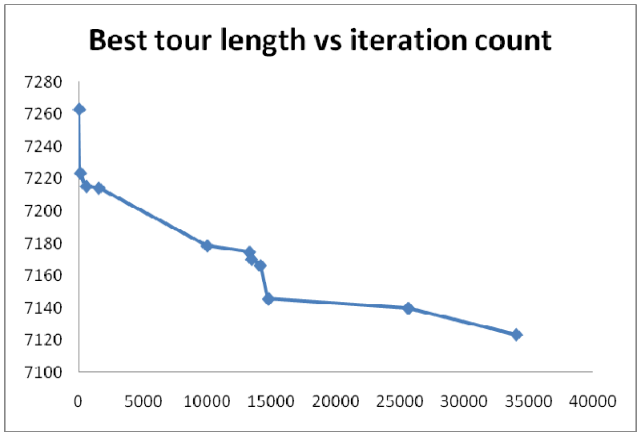

Fig. 2: Behavior of the RACS algorithm for a test case of the Pr439 problem at the first recursive level, applied on a segment of a previous tour.

6. Conclusion

In this paper, a novel ant colony optimization based algorithm to the classical Traveling Salesman Problem has been presented. The general idea underlying the Recursive Ant Colony System is the nature of the ACS paradigm to work more efficiently on problems having lesser number of nodes. The RACS can help to improve the solutions obtained by the ACS algorithms by recursively breaking down the problem and then applying ACS on the smaller parts of the problem, which can improve the exploration efficiency of the algorithm. It gives an optimal strategy for dealing with large TSPs having localised sets of vertices, for which the routes can be optimized by focussing on each set separately and then constructing a globally optimal solution from the local solution obtained. The algorithm can also be used to improve a solution obtained by using any other tour construction algorithm or for getting a better convergence for the Ant Colony meta-heuristics, executing repeatedly to improve solutions. In general, the improvement regarding solution quality is not significant on small problem instances but as the problem size increases the improvement is significant on the majority of large problem instances. However, large problem instances share more similarities with real-world problems and, thus, the RACS paradigm can be more useful on such applications. The application of the algorithm for solving the test cases of the asymmetric TSP depicts the versatility and competence of the algorithm with other Ant Colony algorithms.

For future work, a similar approach can be applied to solve other combinatorial optimization problems where Ant Colony algorithms are in use, such as the quadratic assignment problem [20], vehicle routing problem [21] and the sequential ordering problem [22].

7. References

[1] C.P. Ravikumar. "Solving Large-scale Travelling Salesperson Problems on Parallel Machines". Microprocessors and Microsystems 16(3), pp. 149-158, 1992.

[2] R.G. Bland and D.F. Shallcross. "Large Travelling Salesman Problems arising form Experiments in X-ray Crystallography: A Preliminary Report on Computation". Operations Research Letters 8,pp. 125-128, 1989.

[3] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi, "Optimization by Simulated Annealing", Science, vol. 220, pp. 671-680, 1983.

[4] F. Glover and M. Laguna, "Tabu Search", Kluwer Academic Publishers, 1997.

[5] H.R. Lourenco, O. Martin, and T. Stutzle, "Iterated Local Search", in Handbook of Metaheuristics, ser. International Series in Operations Research & Management Science, F. Glover and G.Kochenberger, Eds., Kluwer Academic Publishers, vol. 57, pp. 321-353, 2002.

[6] J. Holland, "Adaptation in Natural and Artificial Systems", Ann Arbor: University of Michigan Press, 1975.

[7] M.Dorigo, V.Maniezzo, and A.Colorni, "Positive feedback as a search strategy," Diparti-mento di Elettronica, Politecnico di Milano, Italy,mento Tech. Rep. 91-016, 1991.

[8] Dorigo M. & Di Caro G., "AntNet: Distributed Stigmergetic Control for Communication Networks," Journal of Artificial Intelligence Research, Number 9, pp 317-365, 1999.

[9] Dorigo M., Maniezzo V. & Colorni A., "The Ant System: Optimization by a colony of cooperating agents," IEEE Trans. Systems, Man, and Cybernetics- Vol.26,N 1, pp.1-13, 1996

[10] B. Hölldobler and E. O. Wilson, *The Ants*. Berlin: Springer-Verlag, 1990.

[11] G. Reinelt. The Traveling Salesman: Computational Solutions for TSP Applications, volume 840 of LNCS. Springer Verlag, 1994.

[12] Y.H. Song, K.Y. Lee, J.G. Vlachogiannis, Y.H. Lu and I.K. Yu "Ant colony search algorithms in power system optimization," In "Modern Heuristic Optimization Techniques: Theory and Applications to Power Systems," Eds. K.Y. Lee and M.A. El-Sharkawi, IEEE Press Book

[13] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem," SIAM J. Comput., vol. 6, pages 563–581, 1977.

[14] R.M. Karp, “Probabilistic Analysis of Partitioning Algorithms for the Traveling-Salesman in the Plane”, Math. Oper. Res. **2** (1977), 209-224.

[15] A. Rohe, Private Communications (1995).

[16] David S. Johnson and Lyle A. Mcgeoch, Traveling Salesman Problem: A Case Study in Local Optimization.

[17] Christian Nilsson, “Heuristics for the Traveling Salesman Problem”.

[18] Reinelt, G.: TSP-Library (2008). http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[19] Dorian Gaertner and Keith Clark, On Optimal Parameters for Ant Colony Optimization algorithms.

[20] T. Stützle and M. Dorigo, “Aco algorithms for the quadratic assignment problem”, *New Ideas in Optimization*, McGraw-Hill, London, 1999, pp. 3--50.

[21] B. Bullnheimer, R.F. Hartl, and C. Strauss, “Applying the ant system to the vehicle routing problem”, In: Voss S., Martello S., Osman I.H., Roucairol C. (eds.) Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer,Boston, 1999.

[22] L.M. Gambardella and M. Dorigo, “An ant colony system hybridized with a new local search for the sequential ordering problem”, INFORMS Journal on Computing 12 (2000), no. 3, 237--255.