

A Parallel Implementation of Ant Colony Optimization for the Capacited Vehicle Routing Problem

Abstract—Ant Colony Optimization is a new distributed meta-heuristic which is found to be efficient to solve NP-hard combinatorial optimization problems. This paper demonstrates our parallel implementation of Ant Colony Optimization to the well known Capacited Vehicle Routing Problem (CVRP) using OpenMP on shared memory parallel computer architecture. We tested our implementation on some benchmark problem instances. The experiment shows parallel ACO can cope up with large instances of CVRP problems taking less execution time. Parallelization of ant colony optimization to the CVRP problem provides significant speedup and efficiency. Besides, we found our implementation to be near cost-optimal as well as scalable with problem size and system size.

Keywords:

Ant Colony Optimization, Vehicle Routing Problem, Parallel and Distributed Computing, Shared Memory, OpenMP

I. INTRODUCTION

Real ants in search of food create the shortest path from the nest to the food source using some probabilistic rules on pheromone density. Each ant when moves, it deposits some pheromone on its path. Ants follow a path denser in pheromone rather than a lighter one. Also, ants are able to accommodate with changes in their path and if an old path is not usable they are able to find the newer shortest path. If a new obstacle comes in the old path, ants must create a new path. Ants will be divided into two paths around the obstacle, but the shorter one will have greater density in pheromone. Thus, the shorter path will be ultimately followed by the other ants hence creating the shortest path [1].

Many heuristics and meta-heuristics have been developed based on the ant behavior. A meta-heuristic can be defined as a high-level algorithmic approach that can guide many heuristic problems. Ant Colony Optimization (ACO) meta-heuristic [2], Ant Colony System (ACS) [3], MAX-MIN Ant System [4], Multiple Ant Colony System (MACS) [5], AntNet [6], Ant System [7], ABC [8] are some of the meta-heuristics related to the ant behavior. These ant algorithms are extensively used in solving many static and dynamic NP-hard combinatorial optimization problems. The Ant System (AS) was first applied to the travelling salesman problem (TSP) and to the quadratic assignment problem. ACS and MAX-MIN Ant System both have been applied to the symmetric and asymmetric TSP [4]. ABC was applied to routing and load balancing in circuit switched telecommunications networks. AntNet was applied to routing in packet switched telecommunications networks such

as the Internet. AS, ACS, MACS were very recently applied to Vehicle Routing Problems [5], [9].

A. Vehicle Routing Problem

The well known Vehicle Routing Problem is just a variation of the TSP problem. The Vehicle Routing Problem is of different varieties such as CVRP (Capacited VRP), VRPTW (VRP with time windows) and PDPTC (Pickup and Delivery problem under track contention). The simplest version is CVRP where there are n customers who should be supplied goods from a single depot v_0 with a vehicle having total capacity Q . A customer's demand is q_i . Total customer demand can be larger than Q . So the vehicle may need to return to the depot to reload and again go for delivery. Hence, multiple cycles are needed. Here, the objective function is to make total travel time a minimum or the total traversed path length is a minimum. Some variation can be introduced here such as the number of the vehicles is increased or the number of the depots is also increased. The objective function is to find multiple vehicle paths so that total travel time by all vehicles is a minimum or duration of the longest path is a minimum. Sometimes, keeping vehicle count to be a minimum might be important. In VRPTW an additional constraint is added where each customer has a time window $[b, e]$ and the customer must be served within this time. In this report, we will address how ACO is utilized to solve CVRP with single depot using single or multiple vehicles.

In the following sections, firstly relevant research efforts are cited in section 2, the basic idea behind ACO is discussed in section 3. In section 4, the applications of ACO to VRP is described. Also we proposed an algorithm for parallelization in section 4.1. The parallel aspects of ACO are explained in section 5. Section 6 provides conclusions and future works.

II. RELATED WORK

The Vehicle Routing Problem is extensively used in many practical applications such as scheduling of public transports, postal vans, fixing the path of a robot in automated industry, in production engineering for raw materials and intermediate products routing. Therefore, the problem is studied much for many years.

Much research has been done to solve this combinatorial optimization problem. Researchers used Genetic Algorithms, Simulated Annealing, Tabu search, Branch and Bound to solve the VRP problem. However, recent research shows that swarm

based meta-heuristic called ant colony optimization gives better result in many cases. Bullnheimer, Hartl and Strauss first investigated the solution of vehicle routing problem using ant approach [9]. They showed that ACO performs well in VRP than simulated annealing and neural network. However, they found parallel Tabu search provides better solution in respect of path length and execution time over the sequential ACO approach. Recently, Benkner et al. [10] experimented ACO on VRP in cluster or Grid system. They analyze the effectiveness of their method both for fine grained and coarse grained approach. However, to our knowledge, there is no such parallel experiment for VRP on shared memory system.

III. ANT COLONY SYSTEM

In this section, we discuss the Ant Colony System (ACS) [11], [12], [13] that Gamberdella, Dorigo first applied to the travelling salesman problem (TSP). In this paper, ACS has been proposed to solve a CVRP problem having a single depot where both the number of the vehicles and the travel time have to be minimized. ACS associates two measures to each arc of the TSP graph: the closeness/visibility $\eta(r,s)$, and the pheromone trail $\tau(r,s)$. Closeness is defined as the inverse of the arc length. Closeness is a static heuristic value, that is never changed for a particular problem instance. Pheromone trail indicates the amount of pheromone deposited by the ants on an edge. The pheromone trail is dynamically changed by ants at runtime. Therefore, the most important component of ACS is the management of pheromone trails which controls the quality of the solution. The pheromone trail is effectively used to construct new solutions in conjunction with the objective function. Pheromone level in an edge gives a measure of how desirable it is to insert the edge in a solution. Pheromone trails are used for exploration and exploitation. Exploration concerns the probabilistic choice of the components used to construct a solution: a higher probability is given to elements with a strong pheromone trail. Exploitation chooses the component that maximizes a blend of pheromone trail values and heuristic evaluations.

ACS target is to find a shortest tour. In ACS we employ m ants to build the tours in parallel. Initially, each ant is randomly assigned to a starting node and has to build a solution, that is, a complete tour. A tour is built node by node: each ant iteratively adds new nodes until all nodes have been visited.

The probability of adding a new element to the solution space constructed by an ant is a function of the element's heuristic desirability and the pheromone trail.

When ant k is located in node r , it chooses the next node s probabilistically in the set of feasible nodes $J_{k(r)}$ (i.e., the set of nodes that still have to be visited). The probabilistic rule used to construct a tour is given as follows.

$$\begin{aligned} s &= \arg \max_{u \in J_{k(r)}} [\tau(r,u)] * [\eta(r,u)]^\beta \text{ if } q \leq q_0 \\ s &= S \text{ if } q > q_0 \end{aligned} \quad (1)$$

Here q is a random variable uniformly distributed over $[0,1]$. q_0 is tunable parameter where $0 < q_0 < 1$. $J_{k(r)}$ is

the set of unvisited nodes by this ant. β is a user defined parameter that is adjustable. β weighs the relative importance among the heuristic values (i.e. closeness and pheromone trail). q_0 determines the relative importance of exploitation versus exploration: the higher q_0 the higher the probability to use the probabilistic rule described with above equation. s is selected randomly from the set of unvisited nodes, $J_{k(r)}$, using the following probability.

$$p_k(r, s) = \frac{[\tau(r,s)] * [\eta(r,s)]^\beta}{\sum_{u \in J_{k(r)}} [\tau(r,u)] * [\eta(r,u)]^\beta} \quad (2)$$

When $q \leq q_0$, the transition from one node to the next follows the exploitation method using equation 1. It uses the knowledge of the closeness and the pheromone trail to exploit an edge. For, $q > q_0$, the transition follows the exploration method of trying to choose an edge that is unvisited using equation 2.

Once each ant has built a complete solution, the best solution found from the beginning of the trial is used to update the pheromone trails. Then, the process is iterated by starting again m ants until a fixed number of solutions has been generated or a fixed CPU time has elapsed, or no improvement has been achieved during a given number of iterations.

A. Global pheromone updating rule:

After all ants have completed a tour, the edges of the best tour are rewarded. The pheromone density of the edges is increased using the following rule.

$$\begin{aligned} \tau(r, s) &= (1 - \rho) * \tau(r, s) + \rho * \Delta\tau(r, s) \\ \text{where } \Delta\tau(r, s) &= \frac{1}{\text{BestTourLength}} \end{aligned} \quad (3)$$

In this way, the ACS provides emphasis on the probable best solution. This updating rule leads other ants to follow this best path with more probabilistically in subsequent cycles to find the optimal solution. Local pheromone updates are also performed to find other probable solutions.

B. Local pheromone updating rule:

When an ant selects a new vertex s from vertex r within a cycle, pheromone of the edge $\tau(r,s)$ is updated using equation 4. This updating rule is termed as local pheromone update. The purpose of local pheromone update is to reduce the pheromone level so that less ants select the edge in current cycle. In this way more and more paths are explored in a single cycle hence increasing the probability of an extensive path search.

$$\tau(r, s) = (1 - \rho) * \tau(r, s) + \rho * \tau_0 \quad (4)$$

Here, ρ is the pheromone decay parameter and $0 < \rho < 1$ and τ_0 is the initial pheromone level.

IV. FORMULATION OF VRP USING ACO

We can represent CVRP as an undirected graph $G(V,E)$ where V is the set of vertices that represents the customers

and the depot and E is the edge set that represents the routes from one customer to other. Also, m is the number of vehicles having capacity Q , the number of depot is one, and q_i is the demand by one customer. We have to design m vehicles routes and schedules so that the following rules [9] are satisfied.

- Customers will be served exactly by one vehicle and exactly once
- A vehicle must start from depot v_0 and terminate its journey at depot v_0
- The total customer demand under a vehicle route will not exceed the vehicle capacity Q

Procedure Parallel ACO to VRP

Inputs:

V: The list of all vertices

N: The number of iterations

S: The vertices of the traversed path by an ant

Begin

Initialize data structures

Place all ants at the depots or randomly at different customers

For $i \leftarrow 1$ to N **do**

$L \leftarrow V$

For each ant pardo

While $L \neq \text{Empty}$ **do**

Select next vertex v

Local pheromone update

$L \leftarrow L - v$

$S \leftarrow S \cup v$

End while

End For pardo

Find the shortest path S' of all S

For each edge of S'

Global pheromone update

End For

End For

End

End Procedure

Fig. 1. Parallel ACO Algorithm to VRP

A. Proposed algorithm:

To construct routes for the vehicles, some random numbers of ants are applied. However, better solutions are found when the number of ants is equal to the number of customers [9]. Initially, ants are placed at the depot. It is assumed that the ants have the information of the demand of each location. This is somewhat static type of problem, where the demand of a location does not change on the fly. Each ant starts like a vehicle. Ants move from location to location. On their way, they deposit the some scent or pheromone. Each ant also calculates the capacity like real vehicles. When the demand of the customer exceeds the capacity, the ant returns to the depot. The ant chooses the next city in such a way that the

city was never visited by the ant. The ant chooses the city following by the equation 1.

Thus our algorithm follows both the exploitation and exploration. The pheromone concentration also decreases during the iteration to explore the path. The pheromone decrease following by the equation 4. The ants are continuously building the path. After completion of an iteration, the best path is rewarded according to the equation 3. When there are no improvement for certain iterations, the best path is taken as the solution. The pseudo code of the algorithm is given in Figure 1.

V. EXPERIMENTAL RESULTS

The experiments were conducted on an eight processor i686 (Pentium-3) Symmetric Multiprocessor (SMP) Machine. The processor speed is 700.011 MHz. with a cache size of 1024 KB and a total memory space of 6 GB.

We tested the implementation of our algorithm on some CVRP benchmark instances having 32, 64, and 80 customers defined by Augerat et al. [14]. We used C to develop the sequential program and OpenMP library for the multi-threaded parallel program.

The methodologies followed for the experiment are given below

- For every instance of graphs three runs were conducted the runtime is average of these three runs.
- Runs on the same set of inputs were done with number of processors ranging from 1 to 8.
- The experiments were conducted for static scheduling of OpenMP.

Some experimental settings and results are provided below.

A. Parameter Settings

We varied the values of α from 0 to 1 and the values of β from -1 to 0. The values of γ and ρ are varied from 0 to 1. We also varied the value of q_0 from 0 to 1. The values of α , β , γ , ρ are changed with a step 0.125 or 0.1 or 0.25 in different runs and q_0 with a step 0.1. The pseudo code is given in Figure 2.

```

For alpha ← 0 to 1, beta ← -1 to 0
  For gama ← 0 to 1, rho ← 0 to 1
    For q0 ← 0 to 1, step ← 0.1
      Initialization of the problem
      For j ← 0 to max iterations, step ← 1
        Find the solution for current
        parameter set
      End For
    End For
  End For
End For

```

Fig. 2. Parameter Settings

Some good parameter settings are shown in Table 1.

Cust	Param($\alpha, \beta, \gamma = \rho, q_0$)
32	0.5, -0.5, 0.4, 0.1
64	0.25, 0.75, 0.2, 0.2
80	0.8, -0.2, 0.4, 0.2

Table 1: Good Parameter Settings

B. Solution

Both the sequential and parallel implementation of ACS to CVRP provide shortest path very close to the optimal value defined by Augerat et al. for the problem instances. For both sequential and parallel code we got the same shortest path value. Though we have to use variable parameter values and iteration count in two cases. For the same parameter setting the solution for sequential code is worse than the parallel code. The termination point is defined where the shortest path value found by the algorithm does not change over many (200) iterations. The best solutions found using our algorithms are provided in Table 2.

Cust.	Optimal Path(Augerat)	Our Path
32	784	833.99
64	1402	1600.17
80	1764	2041.23

Table 2: Solutions

C. Speedup

The speed up we get is very optimistic. We calculated the speed up using the following equation.

$$Speedup = \frac{Time \text{ Required Using Sequential Code}}{Time \text{ Required Using Parallel Code}}$$

We vary the thread number from 1 to 8. For each case the amount of total work done by the program is equal. For the purpose we kept the number of ants and the number of iterations fixed for each case. To measure the speedup we fixed ant count to be equal to the number of vertex and iteration count to be somewhat around 500-1000. We ran the program for 32 customers. For 2 threads we got the speed up around 1.4. But for 4 threads we get speed up higher than 3. And for 8 threads we get a speed up more than 3.7. This shows the scalability of our program.

Figure 3 shows the speed up result for 32, 64 and 80 customers. In this figure, the speed up for 64 customers is also shown. The speed up for 32, 64 and 80 threads are almost same until 6 threads. After that, the speed up for 8 threads on 80 customers is the most. Also, the speed up for the 64 customers supersedes the speed up for 32 customers. The reason is that, we experiment our algorithm on 8 multiprocessor machine. When there are more work load, the processors can utilize their full capacity. Therefore, it takes relatively less time for larger problems when compared to the sequential version of the same problem. Now it is inferred that the load is balanced to each ant (or thread) evenly.

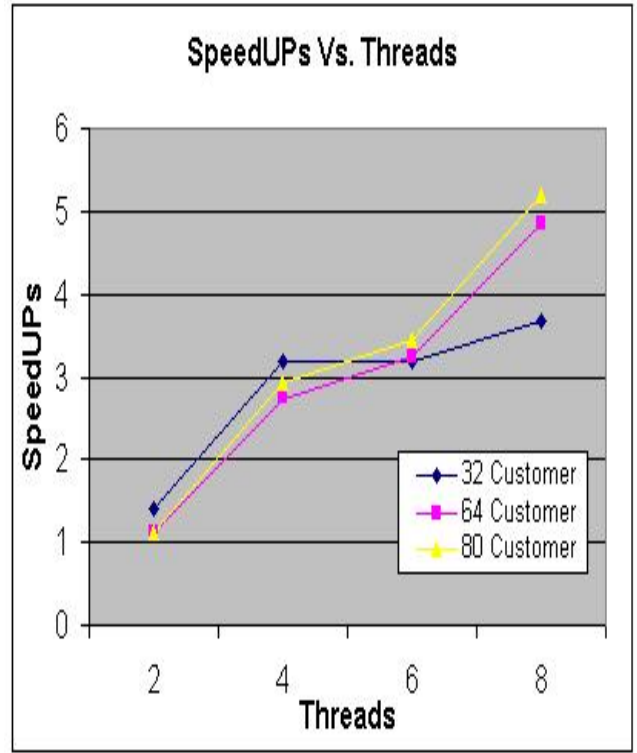


Fig. 3. Speed up results for different threads

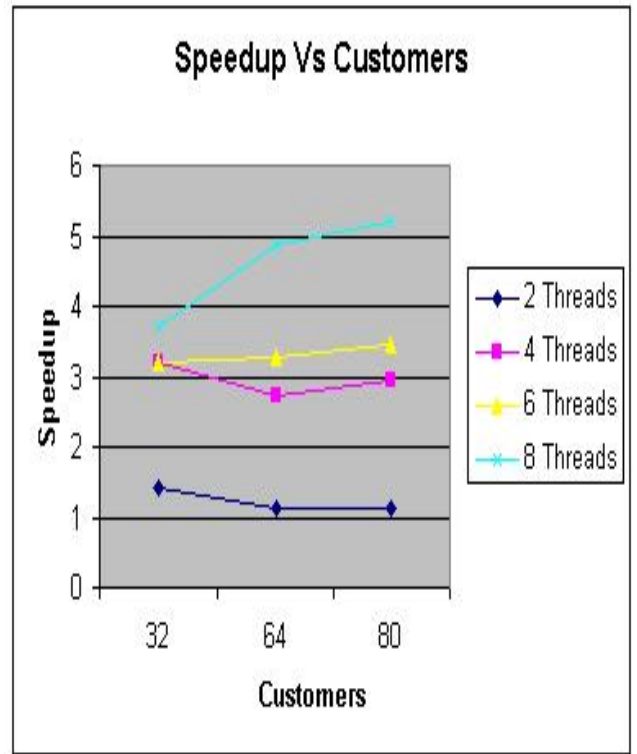


Fig. 4. Speed up results for different number of customers for fixed threads

Figure 4 shows the improvement of efficiency for larger graph. For a graph with 32 customer we got the maximum

speed up for 8 threads is 3.16. On the other hand, for 64 customer, the maximum speed up is 4.89. The interesting result found with 80 nodes graph. The speed up in this case is 5.19. This phenomena clearly indicates that the parallel implementation of the ACO for VRP works better when more works are distributed among the threads and ants. The reason is that when more work is given, the amount of computation increases. This eventually decreases overhead of thread switching time and memory access time. The increase in speedup for increase in data size can be inferred from the fact that the ratio of time spent in execution to the time loss in overhead is bigger for larger data size compared to lower data size.

Customer/Thread	2	4	6	8
32	0.7	0.77	0.53	0.47
64	0.56	0.67	0.54	0.61
80	0.56	0.73	0.57	0.65

Table 3: Efficiency

D. Efficiency

Table 3 shows the efficiency of the code. The efficiency of a parallel code is defined as below

$$efficiency = \frac{Speed\ Up}{No\ of\ Threads}$$

For 32 customers the efficiency is better for 2 and 4 threads. They are more than 0.5. The efficiency for the 64 customers is even better. For all number of threads the efficiency is more than 0.5. This thing also true for the 80 customers.

Cust./Thread	1	2	4	6	8
32	48	34	15	15	13
64	3544	3154	1291	1087	729
80	60712	54322	20718	17616	11696

Table 4: Execution Time in Millisecond

E. Execution Time

From the Table 4, it is clear that the execution time decreases as we increase the number of threads. We can explain the behavior in such a way. When we use only one thread and try to simulate multiple number of ants, the thread is overloaded. When the number of ants equal to the number of threads the load is equally distributed to each thread. Thus, the algorithm works better and proves the efficiency of the parallel implementation.

F. Cost

The processor-time product or cost (or work) of a computation can be defined as follows.

$$cost = (Execution\ Time) * (Total\ Number\ of\ Processors\ Used)$$

The cost of a sequential computation is simply its execution time t_s . The cost of a parallel computation is $t_p * n$ where t_p is the parallel execution time. A cost-optimal parallel algorithm is one in which the cost to solve a problem on a multiprocessor is proportional to the cost (execution time) on a single processor system. The cost of our code is provided in Table 5.

Cust./Thread	1	2	4	6	8
32	48	68	60	90	104
64	3544	6308	5164	6522	5932
80	60712	108644	82872	105696	93568

Table 5: Cost

From Table 5 it is clear that our implementation is not though properly cost optimal, we can say our implementation is near to cost-optimal. For cost-optimal algorithm we know

$$t_p \propto t_s \text{ i.e. } t_p = k * t_s$$

For perfect cost-optimal algorithm the value of k should be the number of processors or threads. In our case k = speedup which is somewhat less than the number of processors or threads. As there is overhead of context switching, and continuous thread spawn and termination the value is reasonable and we can say our implementation is near cost-optimal.

G. Scalability

Scalability indicates that increased problem size can be accommodated with increased system size for a particular architecture and algorithm. Increased system size indicates using more processors. In our case, we see when we increase system size (more processors) the implementation can handle larger problem instances more efficiently. For a fixed problem size with the increase of system size speedup increases. When we increase the problem size then we see if the system is larger we get better speedup and efficiency i.e. larger problem instances are easily accommodated with increased system size. It also indicates that system size increase is logical as it will handle larger problem instances more efficiently. Hence, we can say the architecture and our implementation both are scalable.

For example, we see that when we increase problem size (number of customers) we get speed up \leq lower problem sizes for 2 threads but when system capacity (processor number) is increased we get higher speedup than lower problem sizes. For 64 customers for 2 and 4 threads we get speedup lower than 32 customers but for 6 and 8 threads we get better speedup than 32 customers. For 80 customers for 2 threads we get lower speed up than 32 or 64 customers but for 4, 6, 8 threads we get higher speedup than 32 or 64 customers. This measure indicate that our solution is also scalable. The speedup and efficiency for variable problem size and system size is shown in Table 6 and Table 3 respectively.

Cust/Th	2	4	6	8
32	1.4	3.08	3.18	3.76
64	1.12	2.68	3.24	4.88
80	1.12	2.92	3.42	5.2

Table 6: Speedup

According to Crainic and Toulouse, scalability is a measurement that indicates the capacity of a parallel procedure to provide the same performance level when the problem size and the number of available processors grow proportionally [15]. From the Figure 5 it is clearly seen that our implementation is scalable. We started with 32 customers and 2 threads. Then we doubled the customer number (64 customers) and thread number (4 threads) we get speedup 1.92 times than the speedup of 32 customers which is almost double. Then we increased customer number to 2.5 (80 customer) times and thread number to 3 times (6 threads) we get speed up 2.44 times than 32 customers which is almost 2.5 times. Mentionable, if we would work with 96 customers (3 times of 32 customers) for 6 threads other data reflects that very likely we would get around 3 times speedup than 32 customers. For 5 threads we measured the speedup to be 3.33 i.e. 2.38 times which is almost 2.5 times than 32 customers. This measure also indicates that our implementation is scalable.

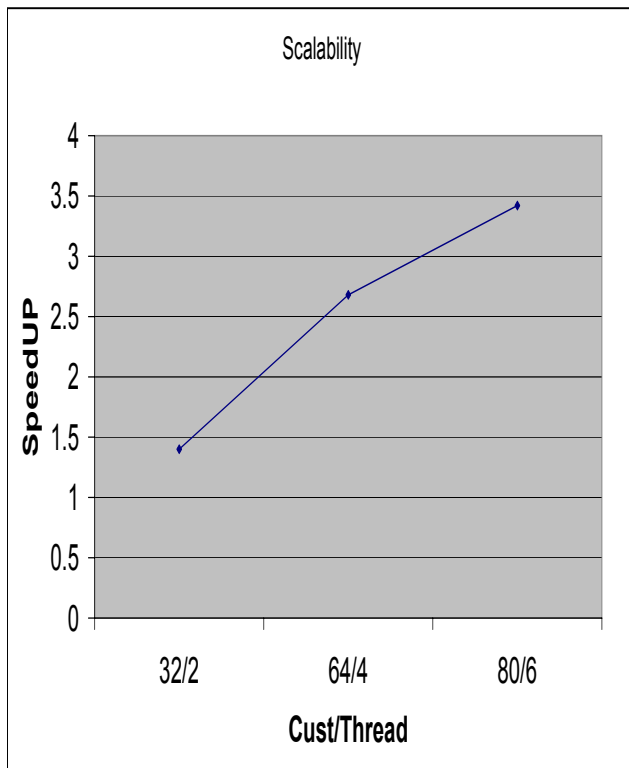


Fig. 5. Scalability Measures

VI. CONCLUSIONS

In this paper, we demonstrate our parallel implementation of an ant algorithm to the Capacitated VRP problem having a single depot. Our parallel OpenMP implementation provides good solution for large instances of CVRP problems taking less execution time. The implementation of our parallel algorithm shows significant speedup and efficiency. Moreover, we found our solution to be near cost-optimal and scalable. However, a detail investigation of the methodologies and the algorithm and parameter values could yield better calibration and would allow further improvements. An efficient local search procedure such as Lin-Kernighan or nearest neighborhood along with a candidate list to select next vertex can improve the solution quality in respect of time and total traversed path length. The parallel algorithm can be extended to other varieties of VRP problems such as CVRP with multiple depots, VRPTW, PDPTC and so on.

REFERENCES

- [1] R. Beckers, J. L. Deneubourg, and S. Goss. Trails and u-turns in the selection of the shortest path by the ant *lasius niger*. *Journal of theoretical biology*, 159:397–415, 1992.
- [2] M. Dorigo and G. D. Caro. The ant colony optimization meta-heuristic. *New Ideas in Optimization McGraw-Hill*, pages 11–32, 1999.
- [3] L. M. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. *International Conference on Machine Learning*, pages 252–260, 1995.
- [4] T. Stützle and H. Hoos. The max-min ant system and local search for the traveling salesman problem. *IEEE 4th International Conference on Evolutionary Computation*, pages 308–313, 1997.
- [5] L. M. Gambardella, E. Taillard, and G. Agazzi. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. *Tech.rep.IDSIA-06-99*, 1999.
- [6] G. D. Caro and M. Dorigo. Antnet: A mobile agents approach to adaptive routing. *Tech. Rep. IRIDIA/97-12, Université Libre de Bruxelles, Belgium*, 1997.
- [7] M. Dorigo, V. Maniezzo, and A. Colomi. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41.
- [8] R. Schoonderwoerd, O. Holland, J. Bruinen, and L. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1997.
- [9] B. Bullnheimer, R. F. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research (Dawid, Feichtinger and Hartl (eds.): Nonlinear Economic Dynamics and Control*, 1999.
- [10] S. Benkner, K. F. Doerner, and R. F. Hartl. Cooperative ant colony optimization on clusters and grids. 2004.
- [11] M. Dorigo L. M. Gambardella. Solving symmetric and asymmetric tsps by ant colonies. *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 622–627, 1996.
- [12] L. M. Gambardella M. Dorigo. Ant colony system: A cooperative learning approach to the traveling. *IEEE Transactions on Evolutionary Computation*, pages 53–66, 1997.
- [13] L. M. Gambardella M. Dorigo. Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–81, 1997.
- [14] Augerat. Cvrp problem instances. <http://neo.lcc.uma.es/radi-aeb/WebVRP/index.html?/Problemstances.html>.
- [15] Crainic T. G. and M. Toulouse. Parallel metaheuristics. In *Kluwer Academic Publishers*, pages 205–251, 1998.