A Parallel Implementation of Ant Colony Optimisation[1]

Marcus Randall
School of Information Technology
Bond University
QLD 4229 Australia
E-mail: mrandall@bond.edu.au

and

Andrew Lewis
School of Computing and Information Technology
Griffith University
QLD 4111 Australia
E-mail: A.lewis@mailbox.gu.edu.au

Ant Colony Optimisation is a relatively new class of meta-heuristic search techniques for optimisation problems. As it is a population based technique that examines numerous solution options at each step step of the algorithm, there are a variety of parallelisation opportunities. In this paper, several parallel decomposition strategies are examined. These techniques are applied to a specific problem, namely the travelling salesman problem, with encouraging speedup and efficiency results.

Key Words: Ant colony optimisation, parallelisation, travelling salesman problem.

1. INTRODUCTION

Ant Colony Optimisation (ACO) is a constructive population based meta-heuristic search technique. As it is a relatively recent addition to the meta-heuristic literature, its development as a standard optimisation tool is still in its infancy. Many aspects require considerable research effort. One of these is the application of parallelisation strategies in order to improve its efficiency, especially for large real world

[1]The authors would like to acknowledge two organisations. The computational experiments were performed on the IBM SP2 operated by the Queensland Parallel Supercomputing Foundation. This research was funded by the Australian Research Council.

problems. This paper classifies some general parallelisation strategies for ACO and describes the application of these to the travelling salesman problem (TSP).

This paper is organised as follows. Section 2 gives a brief overview of ACO meta-heuristics. Section 3 describes some general purpose parallelisation strategies suitable for ACO while Section 4 describes our parallel implementation that solves the TSP. The results are outlined in Section 5 and conclusions are drawn in Section 6.

2. OVERVIEW OF ACO

There are numerous ACO meta-heuristics including Ant System, $\mathcal{MAX} - \mathcal{MIN}$ Ant System and Ant Colony System (ACS) [3]. The ACS search technique is used to implement our test programs as it is representative of these different approaches. ACS can best be described by using the TSP metaphor. Consider a set of cities, with known distances between each pair of cities. The aim of the TSP is to find the shortest path to traverse all cities exactly once and return to the starting city. The ACS paradigm is applied to this problem in the following way. Consider a TSP with N cities. Cities i and j are separated by distance $d(i,j)$. Scatter m virtual ants randomly on these cities ($m \leq N$). In discrete time steps, allow each virtual ant to traverse one edge until all cities are visited. Ants deposit a substance known as *pheromone* to communicate with the colony about the utility of the edges. Denote the accumulated strength of pheromone on edge (i,j) by $\tau(i,j)$.

At the commencement of each time step, Equations 1 and 2 are used to select the next city s for ant k currently at city r. Note: $q \in [0,1]$ is a uniform random number and q_0 is a parameter. To maintain the restriction of unique visitation, ant k is prohibited from selecting a city which it has already visited. The cities which have not yet been visited by ant k are indexed by $J_k(r)$.

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \left\{ \tau(r,s) \cdot [d(r,s)]^{\beta} \right\} & \text{if } q \leq q_0 \\ \text{Equation 2} & \text{otherwise} \end{cases} \tag{1}$$

$$p_k(r,s) = \begin{cases} \frac{\tau(r,s)[d(r,s)]^{\beta}}{\sum_{u \in J_k(r)} \tau(r,u)[d(r,u)]^{\beta}} & \text{if } s \in J_k(r) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

It is typical that the parameter β is negative so that shorter edges are favoured. $\tau(r,s)$ ensures preference is given to links that are well traversed (i.e. have a high pheromone level). Equation 1 is a highly greedy selection technique favouring cities which possess the best combination of short distance and large pheromone levels. Equation 2 balances this by allowing a probabilistic selection of the next city.

The pheromone level on the selected edge is updated according to the local updating rule in Equation 3.

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

Where:

ρ is the local pheromone decay parameter, $0 < \rho < 1$.

τ_0 is the initial amount of pheromone deposited on each of the edges. According to Dorigo and Gambardella [4], a good initial pheromone is $\tau_0 = (NL_{nn})^{-1}$ where L_{nn} is the cost produced by the nearest neighbour heuristic.

Upon conclusion of an iteration (i.e. all ants have constructed a tour), global updating of the pheromone takes place. The edges that compose the best solution to date[2] are rewarded with an increase in their pheromone level. This is expressed in Equation 4.

$$\tau(r,s) \leftarrow (1-\gamma) \cdot \tau(r,s) + \gamma \cdot \Delta\tau(r,s) \tag{4}$$

Where:

$\Delta\tau(r,s)$ is used to enforce the pheromone on the edges of the solution (see Equation 5). L is the length of the best (shortest) tour to date while Q is a constant that is usually set to 100 [5].

$$\Delta\tau(r,s) = \begin{cases} \frac{Q}{L} & \text{if } (r,s) \in \text{ globally best tour} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

γ is the global pheromone decay parameter, $0 < \gamma < 1$.

The basic operation of ACS is described by the pseudocode in Figure 1.

3. GENERAL PARALLELISATION STRATEGIES

Despite the fact that ACO is an inherently parallelisable search technique on a number of levels, little research has been conducted on this aspect apart from Bulnheimer, Kotsis and Strauß [2], Stützle [14] and Michel and Middendorf [9]. The former work is a preliminary investigation of parallelism at the ant (agent) level. However, the authors did not implement their system on a parallel architecture. Hence, it is difficult to determine the efficiency of their parallelisation scheme. Stützle [14] describes the simplest case of parallelisation, that of parallel independent ACO searches that do not interact. Michel and Middendorf [9] describe an island model (adapted from genetic algorithms) in which separate ant colonies exchange trail information.

In this section, five possible parallelisation strategies for ACO meta-heuristics are described. All of these, except for *Parallel Independent Ant Colonies*, are based on the well-known master/slave approach [6] and are hence appropriate for the widely popular MIMD (Multiple Input, Multiple Data) machine architectures [8]. In addition, a standard tool such as the MPI (Message Passing Interface) library can be used to program the ACO engine, ensuring cross platform compatibility. Methods 2, 4 and 5 are new for ACO. In considering different parallel techniques, it must be

[2]This is known as the *global-best* [4] scheme. An *iteration-best* scheme, where the edges of the best solution in the current colony of ants is used, is also possible.

```
Initialise pheromone on all edges;
While (stopping criterion is not met)
  Deposit each ant on a random city such that no two
  ants are placed on the same city;
  For(the number of cities)
    For(each ant)
      Choose the next city to visit according to
      Equation 1;
    End For;
    For(each ant)
      Update the pheromone on each edge according
      to Equation 3;
    End For;
  End For;
  If the best tour from this iteration is better than the
  glabally best tour Then set this is the globally best
  tour;
  Reinforce the pheromone of the edges belonging to the
  globally best tour according to Equation 4;
End While;
Output the globally best tour and cost;
```

FIG. 1. Pseudocode of ACS applied to the TSP.

noted that parallel performance can be degraded when there is large communication overhead between processors. All of the methods assume a distributed rather than shared memory system, as these architectures are more common [6]. However, as ACO systems typically use global memory structures (such as the pheromone matrix), a shared memory machine would mean a lot less communication and a corresponding increase in parallel performance.

3.1. Parallel Independent Ant Colonies

For this approach, a number of sequential ACO searches are run across available processors. Each colony is differentiated on the values of key parameters. While any of the parameters can be varied across the processors, random seed would be the clear choice. The advantage of this method is that no communication is required between the processors. This is a naive approach that can be run as a number of sequential programs on an MIMD machine/cluster of workstations.

3.2. Parallel Interacting Ant Colonies

This approach is similar to the method above, except that at given iterations, an exchange of information between the colonies occurs. The pheromone structure of the 'best' performing colony is copied to the other colonies. The question becomes one of defining the best performing colony, as a number of different measures could be used. The communication cost for this method can be quite high due to the necessity of broadcasting entire pheromone structures, which can be large for many problems.

3.3. Parallel Ants

In this approach, each ant (slave) is assigned a separate processor with which to build its solution. In the case that $m > P$, clustering a number of ants on each processor is required. The master processor is responsible for receiving user input, placing the ants at random solution starting points, performing the global pheromone update and producing the output. It may also act as a slave in order to ensure a more efficient implementation.

This technique has a moderate communication overhead. The largest component is the maintenance of the separate pheromone structures. After the completion of each step of the algorithm, each ant must send an update to each copy of τ in order to satisfy the local pheromone updating rule.

3.4. Parallel Evaluation of Solution Elements

At each step of the algorithm, each ant examines all of the available solution elements before selecting one. This can be quite a computationally expensive operation especially if constraints need to be assessed. As each of the solution elements are independent of one another, they can be evaluated in parallel. Therefore, each slave processor is assigned an equal number of solution elements to evaluate. This is suitable for highly constrained problems.

This approach has been used extensively in the parallelisation of tabu search, see Randall and Abramson [11].

3.5. Parallel Combination of Ants and Evaluation of Solution Elements

Given that enough processors are available, a combination of the previous two strategies is possible. In this case, each ant is assigned an equal number of processors (a group). Within each group, a group master is responsible for constructing the ant's tour and delegating the evaluation of the solution elements to each of the group's slaves. For instance, given ten ants (as commonly used in ACS [4]) and two processors per ant group, this equates to 20 processors. For modern parallel machines, this is not an unreasonable requirement.

4. APPLICATION TO THE TSP

In this paper, one of the aforementioned parallelisation schemes on TSP, namely the *Parallel Ants* scheme is empirically evaluated. Only this scheme is used because of the following reasons. It is believed that the communication overhead for *Parallel Interacting Ant Colonies* will be too large for the TSP. However, for other problems such as the network synthesis problem (see Randall and Tonkes [12]), that have much smaller pheromone structures, this technique would be more appropriate. The *Parallel Evaluation of Solution Elements* technique (and hence the *Parallel Combination of Ants and Evaluation of Solution Elements* technique) is only effective if the cost of the evaluation of an element is high (i.e. the computation is expensive and/or there are numerous and difficult constraints to evaluate), which is not the case for the TSP. Again, the network synthesis problem would benefit from this approach.

Figures 2 and 3 describe the master's and the slaves' activities and communication using pseudocode respectively for the *Parallel Ants* scheme. Note, the terms 'ant' and 'slave' are used interchangably throughout the remainder of this paper. In this algorithm, each processor simulates one ant and maintains its own copy of the pheromone matrix which is incrementally updated throughout the run. This is done to ensure a minimal amount of communication. However, it is anticipated that the bulk of the communication overhead will be in the pheromone updates.

5. COMPUTATIONAL EXPERIENCE

A number of TSP problem instances have been selected with which to test the ACS engine. These problems are from TSPLIB [13] and are given in Table 1.

These problems are run using the set of parameters given in Table 2 as these have been found to give good performance in Dorigo and Gambardella [4] and Dorigo, Maniezzo and Colorni [5]. The computer platform used to perform the experiments is an IBM SP2 consisting of 18 RS6000 model 590 processors with a peak performance of 266 MFLOPS per node. At most eight dedicated processors are available for parallel computation on this machine.

The guidelines for reporting parallel experiments as outlined in Barr and Hickman [1] are followed. The most common measure of effectiveness of a parallel algorithm is given by *speedup*. Speedup is defined by Equation 6.

```
Get user parameters(β, q_0, γ, ρ, seed);
Broadcast (β, q_0, γ, ρ, seed) to each ant;
L_nn =Calculate the nearest neighbour cost;
τ_0 = (N L_nn)^{-1};
Broadcast τ_0 to each ant;
Broadcast the d matrix and N to each ant;
While (termination condition not met)
  Deposit each ant on a random city such that no two
  ants are placed on the same city;
  Send each initial city to each ant;
  For (each city)
    Receive each ant's next city and add to the colony solution;
    Update the pheromone matrix using the local update rule;
    Broadcast m pheromone updates (i, j, τ_{i,j}) to each ant;
  End For;
  Receive the cost of each ant's solution;
  iteration_best_cost =Determine the best solution
  cost from each the current colony;
  If (iteration_best_cost < best_cost)
    best_cost = iteration_best_cost;
  End If;
  Update the pheromone matrix using the global update rule;
  Broadcast N pheromone updates to each ant;
  Determine if the termination condition is met and broadcast
  to each ant;
End While;
End.
```

FIG. 2. The pseudocode for the master processor applied to the TSP for the *Parallel Ants* strategy.

```
Receive (β, q0, γ, ρ, seed) from the master;
Receive τ0 from the master;
Receive the d matrix and N from the master;
Initialise the pheromone matrix with τ0;
While (termination condition is not met)
  initial_city = city =receive the initial city
  from the master;
  For (each city)
    next_city =choose the next city according to
    Equation 1;
    Send next_city to the master;
    cost = cost + d_{city,next_city};
    city = next_city;
  End For;
  cost = cost + d_{next_city,initial_city};
  Send cost to the master;
  Receive the pheromone update from the master;
  Receive the termination information signal from
  the master;
End While;
End.
```

FIG. 3. The pseudocode for the slave processors applied to the TSP for the *Parallel Ants* strategy.

TABLE 1
Problem instances used in this study.

Name	Size (cities)	Best-Known Cost
gr24	24	1272
st70	70	675
kroA100	100	21282
kroA200	200	29368
lin318	318	42029
pcb442	442	50778
rat575	575	6773
d657	657	48912

TABLE 2
Parameter settings used in this study.

Parameter	Value
β	-2
γ	0.1
ρ	0.1
m	2 ... 8
q_0	0.9
iterations	1000

$$speedup = \frac{\begin{array}{c}\text{Time to solve a problem with the fastest serial} \\ \text{code on a specific parallel computer}\end{array}}{\begin{array}{c}\text{Time to solve the same problem with the parallel} \\ \text{code using } P \text{ processors on the same computer}\end{array}} \tag{6}$$

According to Barr and Hickman, average values should not be used in Equation 6 and would require a new definition of speedup. As a result, only one seed per problem instance (and processor grouping for Method 4 and 5) is used. The numerator of Equation 6 is measured by CPU time whereas the denominator is measured by wall clock time. The efficiency of the parallel code is computed by Equation 7.

$$efficiency = \frac{speedup}{P} \tag{7}$$

The results are outlined in Table 3. For the small problems, parallelisation is ineffective and counterproductive as the amount of communication means that the real time spent by the parallel code far exceeds the serial code. However, there is a steady (near linear) increase in the parallel efficiency when more processors are

added and the problem size is increased. Problems having over 200 cities receive benefit from the parallelisation scheme.

TABLE 3
Parallel speedup and efficiency results. The first entry in each cell is speedup while the second is efficiency.

Problem	P						
	2	3	4	5	6	7	8
gr24	0.08	0.06	0.07	0.07	0.07	0.06	0.06
	0.04	0.02	0.02	0.01	0.01	0.01	0.01
st70	0.27	0.21	0.23	0.24	0.23	0.21	0.21
	0.13	0.07	0.06	0.05	0.04	0.03	0.03
kroA100	0.41	0.32	0.37	0.38	0.37	0.34	0.33
	0.2	0.11	0.09	0.08	0.06	0.05	0.04
kroA200	0.82	0.84	0.85	0.92	0.92	0.91	0.9
	0.41	0.28	0.21	0.18	0.15	0.13	0.11
lin318	1.2	1.44	1.44	1.59	1.61	1.53	1.58
	0.6	0.48	0.36	0.32	0.27	0.22	0.2
pcb442	1.42	1.62	1.93	2.18	2.31	2.31	2.35
	0.71	0.54	0.48	0.44	0.38	0.33	0.29
rat575	1.56	1.78	2.1	2.55	2.77	3.02	3.08
	0.78	0.59	0.52	0.51	0.46	0.43	0.38
d657	1.67	1.95	2.32	2.89	3.25	3.29	3.3
	0.83	0.65	0.58	0.58	0.54	0.47	0.41

Figure 4 graphically shows the speedup using six processors. While by many measures speedup is rather poor, the graph and Table 3 show that speedup > 1 is achieved for problems lin318 and above, with a maximum speedup of 3.3. Speedup and efficiency increase as problem size increases. Hence for large problems, this parallelisation strategy could be used to decrease the amount of real time required.

6. CONCLUSIONS

The most appropriate parallelisation technique is ultimately dependent upon the nature of the problem being solved. For problems in which most computational effort is concentrated in the evaluation of the solution elements, methods 3,4 and 5 are appropriate. Problems like the TSP in which each of the solution elements may be easy to compute, yet each solution contains many such elements, method 3 would be suitable. For those problem having a small pheromone structure, method 2 is a viable alternative. The above rules are only a guide to the parallelisation of ACO meta-heuristics and as such, a more formal and generic set should be investigated.

In this paper, the *Parallel Ants* scheme in which ants construct tours in parallel has been evaluated. A master ant is used to co-ordinate the activities of the colony. This scheme is conceptually simple and suitable for the popular MPI model on MIMD architectures. The results showed that acceptable speedup and efficiency can be achieved for larger problems ($N > 200$). However, one of the disadvantages

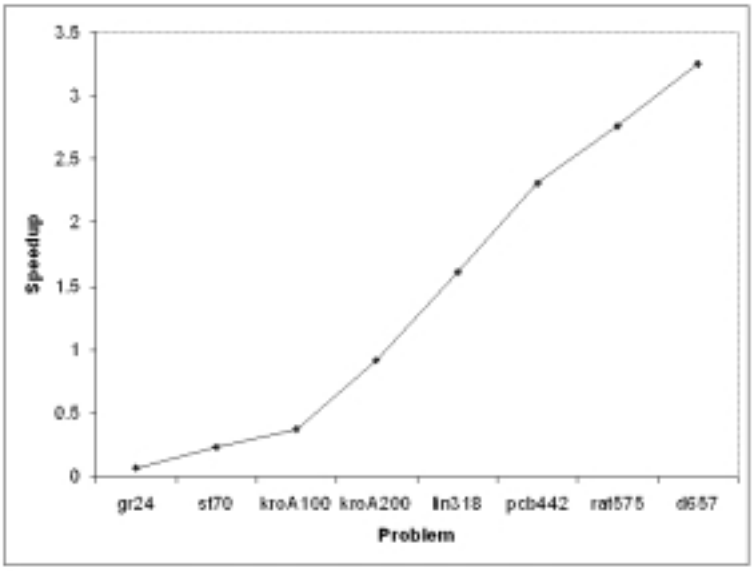

FIG. 4. Speedup on each problem using six processors.

to this scheme is the large amount of communication required to maintain the pheromone matrix.

Parallel efficiency would improve if the algorithm incorporated a larger parallelisable component. For instance, each ant could add a local search phase at the end of the construction of their tour. Another way is to use a shared memory computer. Our future work will concentrate on minimising (both absolutely and relatively) the amount and frequency of this communication. We are also investigating candidate list strategies [10] in order to reduce the number of solution elements examined by each ant at each step. Candidate list combined with effective parallelisation strategies should yield effective ACO problem solvers.

At the present time, our parallel code only allows for one ant per processor. In future versions, the number of ants will be scaled to the number of available processors. For instance, if there are ten ants but only five available processors, each processor will simulate two ants.

REFERENCES

1. Barr, R. and Hickman, B. (1993) "Reporting Computational Experiments with Parallel Algorithms: Issues, Measures and Experts' Opinions", ORSA Journal on Computing, 5, pp. 2-18.

2. Bullnheimer, B., Kotsis, G. and Strauß, C. (1998) "Parallelization Strategies for the Ant System", In De Leone, R., Murli, A., Pardalos, P. and Toraldo, G. (eds.), High Performance Algorithms and Software in Nonlinear Optimization; series: Applied Optimization, 24, Kluwer: Dordrecht, pp. 87-100.

3. Dorigo, M. and Di Caro, G. (1999) "The Ant Colony Meta-heuristic", in New Ideas in Optimization, Corne, D., Dorigo, M. and Golver, F. (eds.), McGraw-Hill, pp. 11-32.

4. Dorigo, M. and Gambardella, L. (1997) "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", IEEE Transactions on Evolutionary Computing.

5. Dorigo, M., Maniezzo, V. and Colorni, A. (1996) "The Ant System: Optimization by a Colony of Cooperating Agents", IEEE Transactions on Systems, Man and Cybernetics - Part B, 26, pp. 29-41.
6. Foster, I. (1994) Designing and Building Parallel Programs, Addison Wesley: Reading, MA, 381 pages.
7. Glover, F. and Laguna, M. Tabu Search, Kluwer Academic Publishers: Boston, MA, 442 pages.
8. Kumar, V., Grama, A., Gupta, A. and Karypis, G. (1994) Introduction to Parallel Computing, Benjamin Cummings: Reading, MA, 597 pages.
9. Michel, R. and Middendorf, M. (1998) "An Island Based Ant System with Lookahead for the Shortest Common Subsequence Problem", Proceedings of the fifth International Conference on Parallel Problem Solving from Nature, 1498, Springer Verlag, pp. 692-708.
10. Randall, M. and Montgomery, J.(2001) "Candidate Set Strategies for Ant Colony Optimisation", Working paper.
11. Randall, M. and Abramson, D. (1999) "A General Parallel Tabu Search Algorithm for Combinatorial Optimisation Problems", Proceedings of the 6^{th} Australasian Conference on Parallel and Real Time Systems, Cheng, W. and Sajeev, A. (eds), Springer-Verlag, pp. 68-79.
12. Randall, M. and Tonkes, E. (2000) "Solving Network Synthesis Problems using Ant Colony Optimisation", Lecture Notes in Artificial Intelligence, 2070, Monostori, L., Vancza, J. and Ali, M. (eds), Springer Verlag: Berlin, pp. 1-10
13. Reinelt, G. (1991) "TSPLIB - A Traveling Salesman Problem Library", ORSA Journal on Computing, **3**, pp. 376-384.
14. Stützle, T. (1998) "Parallelization Strategies for Ant Colony Optimization", Proceedings of Parallel Problem Solving from Nature, Eileen, A., Bäck, T., Schoenauer, M. and Schwefel, H. (eds.), Springer Verlag, 1498, Lecture Notes in Computer Science, pp. 722-741.