Journal of Parallel and Distributed Computing **62**, 1421–1432 (2002)
doi:10.1006/jpdc.2002.1854

A Parallel Implementation of Ant Colony Optimization

Marcus Randall[1,2]

School of Information Technology, Bond University, Qld 4229, Australia
E-mail: mrandall@bond.edu.au

and

Andrew Lewis[1]

School of Computing and Information Technology, Griffith University, Qld 4111, Australia
E-mail: a.lewis@mailbox.gu.edu.au

Received March 29, 2000; accepted January 29, 2002

Ant Colony Optimization is a relatively new class of meta-heuristic search techniques for optimization problems. As it is a population-based technique that examines numerous solution options at each step of the algorithm, there are a variety of parallelization 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 optimization; parallelization; travelling salesman problem.

1. INTRODUCTION

Ant Colony Optimization (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 optimization tool is still in its infancy. Many aspects require considerable research effort. One of these is the application of parallelization strategies in order to improve its efficiency, especially for large real-world problems. This paper classifies some general parallelization strategies for ACO and describes the application of these to the travelling salesman problem (TSP).

[1]The authors acknowledge two organizations. 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.

[2]To whom correspondence should be addressed.

0743-7315/02 $35.00

This paper is organized as follows. Section 2 gives a brief overview of ACO meta-heuristics. Section 3 describes some general purpose parallelization 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) [4]. 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 ($m \leqslant N$). 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, Eqs. (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 $J_k(r)$:

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

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

It is typical that the parameter β is negative so that shorter edges are favored. $\tau(r, s)$ 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 favoring 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 Eq. (3).

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

where ρ is the local pheromone decay parameter, $0 < \rho < 1$ and τ_0 is the initial amount of pheromone deposited on each of the edges. According to Dorigo and Gambardella [5], a good initial pheromone is $\tau_0 = (NL_{\text{nn}})^{-1}$ where L_{nn} is the cost produced by the nearest-neighbor 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[3] are rewarded with an increase in their pheromone level. This is expressed in

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

where $\Delta\tau(r,s)$ is used to enforce the pheromone on the edges of the solution (see Eq. (5)). L is the length of the best (shortest) tour to date while Q is a problem-dependent constant that is usually set to 100 [6]:

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

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

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

3. GENERAL PARALLELIZATION STRATEGIES

Despite the fact that ACO is an inherently parallelizable search technique on a number of levels, little research has been conducted on this aspect apart from Bullnheimer *et al*. [3], Stützle [17] and Michel and Middendorf [11]. 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 parallelization scheme. Stützle [17] describes the simplest case of parallelization, that of parallel independent ACO searches that do not interact. Michel and Middendorf [11] describe an island model (adapted from genetic algorithms) in which separate ant colonies exchange trail information.

When considering general parallelization strategies, parallelism can be exploited at one, or more, of several scales. At the largest scale, entire searches can be performed concurrently, as described by Stützle [17] and outlined in the first two possible parallelization strategies for ACO meta-heuristics described below. Alternatively, parallelism can sometimes be found in the evaluation of solution elements, especially, when this evaluation is computationally expensive. For the TSP used as a basis for the investigations described in this paper, solution element evaluation can be considered trivial and so parallelization at this scale is inappropriate. All of the remaining strategies described seek to exploit parallelism within the meta-heuristic algorithm.

All of these, except for *Parallel Independent Ant Colonies*, are based on the well-known master/slave approach [7] and are hence appropriate for the widely popular multiple input, multiple data (MIMD) machine architectures [10]. In addition, a standard tool such as the message passing interface (MPI) library can be used to program the ACO engine, ensuring cross platform compatibility. Methods 2, 4 and 5

[3] This is known as the *global-best* [5] 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.

are new for ACO. In considering different parallel techniques, it must be 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 [7]. 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 potential 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 parallelization of tabu search, see Randall and Abramson [12].

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 10 ants (as commonly used in ACS [5]) 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 parallelization 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 [15]), 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 interchangeably 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 [16] and are given in Table 1.

```
Get user parameters(β, q0, γ, ρ, seed);
Broadcast (β, q0, γ, ρ, seed) to each ant;
L_nn =Calculate the nearest neighbour cost;
τ0 = (N L_nn)^-1;
Broadcast τ0 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_ij) 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 fr
  the master;
End While;
End.
```

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

These problems are run using the set of parameters given in Table 2 as these have been found to give good performance in [5,6]. 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 [2] are followed. The most common measure of effectiveness of a parallel algorithm is given by

TABLE 1

Problem Instances Used in This Study

Name	Size (cities)	Best-known cost
gr24	24	1272
st70	70	675
kroA100	100	21,282
kroA200	200	29,368
lin318	318	42,029
pcb442	442	50,778
rat575	575	6773
d657	657	48,912

TABLE 2

Parameter Settings Used in This Study

Parameter	Value
β	-2
γ	0.1
ρ	0.1
m	$2 \cdots 8$
Q	100
q_0	0.9
Iterations	1000

speedup. Speedup is defined by

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

According to Barr and Hickman [2], average values should not be used in Eq. (6) and would require a new definition of speedup. As a result, only one seed per problem instance (and processor grouping for methods 4 and 5) is used. The numerator of Eq. (6) is measured by CPU time whereas the denominator is measured by wall clock time. The efficiency of the parallel code is computed

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

The results[4] are outlined in Table 3. For the small problems, parallelization 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 parallelization scheme.

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 parallelization strategy could be used to decrease the amount of real time required.

To further investigate the parallel behavior, the experimental serial fraction was calculated for the four larger problems that obtained benefit from parallelization, after the method of Karp and Flatt [9]. This measure is given by

$$\text{Experimental serial fraction} = \frac{\frac{1}{\text{speedup}} - \frac{1}{P}}{1 - \frac{1}{P}}. \tag{8}$$

[4] The performance of the ACS engine in terms of generating short tour lengths is found in [14].

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

The results for each problem, graphed against the number of processors used, are shown in Fig. 5. It can be seen that the experimental serial fraction decreases with problem size, implying improved parallel performance. Each problem appears to reach a minimum using approximately six processors and then increase. However,

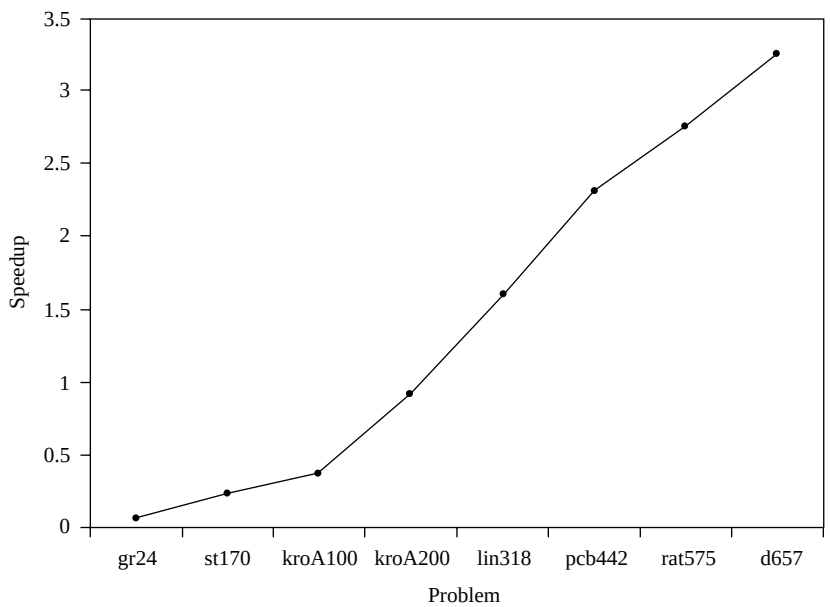

FIG. 4. Speedup on each problem using six processors.

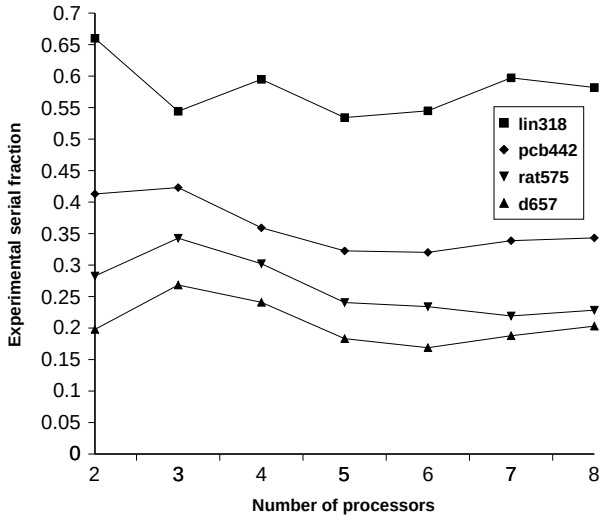

FIG. 5. Experimental serial fraction on each problem.

the experimental results obtained do not allow characterization of the scaling behavior with larger numbers of processors.

The significance of this measure can be seen when the experimental serial fraction is used to derive the theoretical maximum speedup using Amdahl's Law [1]. For code with a serial fraction of s, this states the speedup approaches an upper limit of $\frac{1}{s}$. The theoretical maximum speedups for the four problems are shown in Fig. 6. While increasing with problem size, the rate of increase slows for the largest problem. Further, investigation with even larger problems is necessary to determine if this decrease continues.

In addition, from Table 3 it may be seen that the problems tested have obtained a significant fraction of the theoretical maximum speedup using quite modest numbers

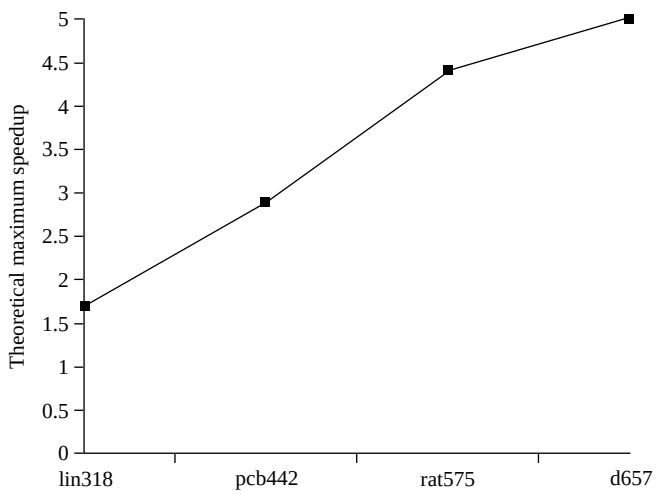

FIG. 6. Theoretical maximum speedup on each problem.

of processors. This suggests scaling these problems to larger numbers of processors will yield little further benefit.

6. CONCLUSIONS

The most appropriate parallelization 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 3–5 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 parallelization 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 coordinate 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 to this scheme is the large amount of communication required to maintain the pheromone matrix. Further investigation is required to fully characterize the scalability of the scheme with larger problems and greater numbers of processors.

Parallel efficiency would improve if the algorithm incorporated a larger parallelizable 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 minimizing (both absolutely and relatively) the amount and frequency of this communication. We are also investigating candidate list strategies [13] in order to reduce the number of solution elements examined by each ant at each step. The candidate list technique combined with effective parallelization 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 10 ants but only five available processors, each processor will simulate two ants.

REFERENCES

1. G. Amdahl, Validity of the single processor approach to achieving large scale computer capabilities, *in* "Proceedings of the AFIPS Spring Joint Computer Conference," Vol. 30, pp. 483–485, Atlantic City, NJ, 1967.
2. R. Barr and B. Hickman, Reporting computational experiments with parallel algorithms: Issues, measures and experts' opinions, *ORSA J. Comput.* **5** (1993), 2–18.
3. B. Bullnheimer, G. Kotsis, and C. Strauß, Parallelization strategies for the ant system, *in* "High Performance Algorithms and Software in Nonlinear Optimization; Series: Applied Optimization," R. De Leone, A. Murli, P. Pardalos, and G. Toraldo (Eds.), Vol. 24, pp. 87–100, Kluwer, Dordrecht, 1998.

4. M. Dorigo and G. Di Caro, The ant colony meta-heuristic, *in* "New Ideas in Optimization," D. Corne, M. Dorigo, and F. Golver (Eds.), pp. 11–32, McGraw-Hill, New York, 1999.
5. M. Dorigo and L. Gambardella, Ant colony system: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Trans. Evolut. Comput.* **1** (1997), 53–66.
6. M. Dorigo, V. Maniezzo, and A. Colorni, The ant system: Optimization by a Colony of Cooperating Agents, *IEEE Trans. Systems Man Cybernet.*—Part B **26** (1996), 29–41.
7. I. Foster, "Designing and Building Parallel Programs," 381pp, Addison–Weslley, Reading, MA, 1994.
8. Deleted in proof.
9. A. Karp and H. Flatt, Measuring parallel processor performance, *Comm. ACM* **33** (1990), 539–543.
10. V. Kumar, A. Grama, A. Gupta, and G. Karypis, "Introduction to Parallel Computing," 597 pp., Benjamin Cummings, Reading, MA, 1994.
11. R. Michel and M. Middendorf, An island based Ant System with lookahead for the shortest common subsequence problem, *in* "Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature," Vol. 1498, pp. 692–708, Springer-Verlag, Berlin, 1998.
12. M. Randall and D. Abramson, A General Parallel Tabu Search Algorithm for Combinatorial Optimization Problems, *in* "Proceedings of the Sixth Australasian Conference on Parallel and Real Time Systems," W. Cheng and A. Sajeev (Eds.), pp. 68–79, Springer-Verlag, Berlin, 1999.
13. M. Randall and J. Montgomery, "Candidate Set Strategies for Ant Colony Optimization," Technical Report, TR02-04, School of Information Technology, Bond University, 2002.
14. M. Randall and E. Tonkes, Intensification and diversification strategies in ant colony system, *Complexity Internat.* (2000), submitted. Available online at http://life.cs.edu.au/ci/sub07/randal01.
15. M. Randall and E. Tonkes, "Solving Network Synthesis Problems using Ant Colony Optimization," L. Monostori, J. Vancza, and M. Ali (Eds.), Lecture Notes in Artificial Intelligence, Vol. 2070, pp. 1–10, Springer-Verlag, Berlin, 2001.
16. G. Reinelt, TSPLIB—A traveling salesman problem library, *ORSA J. Comput.* **3** (1991), 376–384.
17. T. Stützle, Parallelization strategies for ant colony optimization, *in* "Proceedings of Parallel Problem Solving from Nature," A. Eileen, T. Bäck, M. Schoenauer, and H. Schwefel (Eds.), Lecture Notes in Computer Science, Vol. 1498, pp. 722–741, Springer-Verlag, Berlin, 1998.

ANDREW LEWIS is Manager, Research Computing Services and Manager of the Research Centre for Modelling and Computation at Griffith University. He is currently Acting CEO of the Queensland Parallel Supercomputing Foundation and has been involved in high performance computing and industrial applied research since 1985.

Andrew has publications in high performance computing, optimization and computational fluid dynamics. His current research interests are in parallel optimization algorithms and their application to numerical simulations.