# Candidate Set Strategies for Ant Colony Optimisation

Marcus Randall and James Montgomery*

School of Information Technology
Bond University, QLD 4229
{mrandall, jmontgom}@bond.edu.au

**Abstract.** Ant Colony Optimisation based solvers systematically scan the set of possible solution elements before choosing a particular one. Hence, the computational time required for each step of the algorithm can be large. One way to overcome this is to limit the number of element choices to a sensible subset, or candidate set. This paper describes some novel generic candidate set strategies and tests these on the travelling salesman and car sequencing problems. The results show that the use of candidate sets helps to find competitive solutions to the test problems in a relatively short amount of time.

*Keywords:* Ant colony optimisation, candidate set, travelling salesman problem, car sequencing problem.

## 1 Introduction

The Ant Colony Optimisation (ACO) [1] meta-heuristic is a relatively new optimisation technique based on the mechanics of natural ant colonies. It has been applied extensively to benchmark problems such as the Travelling Salesman Problem (TSP), the job sequencing problem and the Quadratic Assignment Problem (QAP). In addition, work on more complex problems that have difficult constraints in such areas as transportation and telecommunications, has been undertaken [1]. Since ACO is a constructive meta-heuristic, at each step ants consider the entire set of possible elements before choosing just one. Hence, the vast majority of an ant algorithm's runtime is devoted to evaluating the utility of reachable elements and so ACO techniques can suffer from long runtimes if attention is not paid to constructing appropriate subsets of elements from which to choose. There has been little work done in this area, despite the fact that this can potentially improve the efficiency of ACO, especially for large real world problems. The way that this is achieved is via *candidate set strategies*.

Candidate set strategies have traditionally only been used as part of a local search procedure applied to the solutions generated by ACO. However, the strategies developed for local search heuristics such as 2-opt and 3-opt are inappropriate for use in the construction phase of the ACO algorithm, and it is only

* This author is a PhD scholar supported by an Australian Postgraduate Award.

in later improvements of Ant Colony System (ACS) that candidate set strategies were applied as part of the construction process [2,3]. The most common candidate set used for the TSP is *nearest neighbour*, in which a set of the $k$ nearest cities is maintained for each city. Ants select from this set first and only if there are no feasible candidates are the remaining cities considered. This approach has been particularly useful on larger problems (more than 1500 cities) [2]. In some instances, maintaining sets of less than 10 cities can be sufficient to contain all the links in the optimal solution [4]. Bullnheimer, Hartl and Strauß [5] also use a nearest neighbour candidate set their ant system for the vehicle routing problem. Another static approach for geometric problems is to create a candidate set based on the Delaunay graph, augmented with extra edges to provide sufficient candidates [4].

Both the nearest neighbour and Delaunay graph candidate set approaches can be easily generalised for problems in which each element has a relationship with each other element, such as the TSP. However, for problems that do not exhibit strong relationships between elements, such as in the QAP where facilities relate to locations but not to other facilities, these techniques are difficult to apply.

Each of these techniques uses static candidate sets generated *a priori* (i.e. candidate sets that are derived before, and not updated or changed during, the application of the ACO meta-heuristic). In contrast, *dynamic* candidate set strategies require the sets to be regenerated throughout the search process. Although used routinely in iterative meta-heuristics like Tabu Search (TS) [6], their use in ant algorithms has only been suggested by Stützle and Hoos [7]. As much of the power of ACO comes from the use of adaptive memory (pheromone trails), it is likely that using dynamic candidate set strategies will lead to further improvements in both solution quality and computational runtime. This paper outlines generic dynamic candidate set strategies for a wide variety of common combinatorial optimisation problems. In addition, we test some appropriate generic strategies on the TSP and the car sequencing problem (CSP) [8]. An extended version of this paper is also available [9].

This paper is organised as follows. Section 2 gives a description of some generic candidate set strategies. Section 3 outlines the computational experiments and Section 4 has the concluding remarks.

## 2 Generic Candidate Set Strategies

Candidate sets strategies can be broadly divided into two approaches: *static*, in which candidate sets are generated *a priori* and used without change throughout the search process, and *dynamic*, in which candidates sets are regenerated during the search. Static approaches are more problem specific and suitable for simpler problems such as the TSP. Dynamic approaches are more easily generalised across different problem types and hence, their development and refinement is necessary to solve complex problems.

The TS meta-heuristic was the first to make use of dynamic candidate list strategies [6]. Greedy Randomised Adaptive Search Procedures (GRASPs) [10], another constructive approach, also use a type of dynamic candidate set that is highly similar to the first strategy described below. We describe how the TS strategy of *elite candidate set* and a new general purpose strategy, *evolving set* can be applied to ACO. Both of these approaches maintain separate candidate sets for each element in the solution, as in the static approaches used previously in ACO.

1. *Elite Candidate Set.* Initially, the candidate set is established by considering all possible elements and selecting the best $k$, where $k$ is the size of the set, based on their probability values (see Dorigo and Gambardella [2]). This set is then used for the next $l$ iterations of the algorithm. The rationale of this approach is that a good element now is likely to be a good element in the future.
2. *Evolving Set.* This is similar to the Elite candidate set strategy and follows an important aspect of the TS process. Elements that give low probability values are eliminated temporarily from the search process. These elements are given a "tabu tenure" [6] in a tabu list mechanism. This means that for a certain number of iterations, the element is not part of the candidate set. After this time has elapsed, it is reinstated to the candidate set. The threshold for establishing which elements are tabu can be varied throughout the search process depending on the quality of solutions being produced.

## 3 Computational Experience

Two problem classes were used to test the generic candidate set strategies, the TSP and the CSP. The CSP is a common problem in the car manufacturing industry [8]. In this problem, a number of different car models are sequenced on an assembly line. The objective is to separate cars of the same model type as much as possible in order to evenly distribute the manufacturing workload. The TSP problem instances, from TSPLIB, are gr24, hk48, eil51, st70, eil76, kroA100, d198, lin318, pcb442 and att532. The CSP problem instances are n20t1, n20t5, n40t1, n40t5, n60t1, n60t5, n80t1 and n80t5, and are available online at `http://www.it.bond.edu.au/randall/carseq.tar`.

Our experiments are based on the ACS meta-heuristic. The computing platform used to perform the experiments is a 550 MHz Linux machine. Each problem instance is run across 20 random seeds and consists of 3000 iterations. The ACS parameter values used are: $\beta = -2$, global pheromone decay $\alpha = 0.1$, local pheromone decay $\rho = 0.1$, number of ants $m = 10$, $q_0 = 0.9$.

For both problem types, a control strategy and an ACS with a static candidate set were run in order evaluate the performance of the generic dynamic strategies. The control ACS is simply ACS without any candidate set features. The static set strategy for the TSP and CSP is nearest neighbour. For the CSP, the separation distance between each pair of cars is calculated to produce the nearest neighbour static set. In our experiments, we set $k = 10$ for both TSP and CSP.

## 3.1 Code Implementation

This section describes the mechanics of implementing the candidate set strategies within the ACS framework. In particular, the heuristic for establishing and varying quality thresholds is described and problem specific details are given.

**Elite Candidate Set** Elite candidate set as described in the TS literature regenerates its candidate set after a predetermined number of iterations or if the quality of elements in the set falls below a critical level. Our implementation uses the former strategy, as it stores element quality at the time it generates the candidate set. This makes it difficult to judge whether the *current* quality of elements has dropped sufficiently to necessitate the regeneration of the set. However, initial testing has shown that our method ensures that elite candidate set runs quickly, while having minimal impact on the cost of solutions generated. A candidate set may persist across iterations. Elite candidate set has two control parameters: the size of the set (expressed as a constant number of elements) and the refresh frequency (expressed as a (fractional) number of iterations). The values used in the experiments are a set size of 10 and a refresh frequency of 0.5 iterations.

**Evolving Set** At each step, the Evolving set strategy examines only those elements that are reachable by an ant during that step to determine their tabu status. Elements whose tabu tenure has expired cannot be immediately reincluded on the tabu list. Elements can, and in these experiments were, placed on the tabu list for more than one iteration. Hence, Evolving set can serve to focus the search over a number of iterations, rather than just within a single iteration. In addition to the tabu threshold, which is adjusted by the algorithm, Evolving set has only one parameter, the tabu tenure. For these experiments this was set at 15 iterations. Tabu tenures smaller than one iteration were used initially but found to be less effective.

A simple heuristic has been developed for adjusting the tabu threshold periodically between preset upper and lower bounds. An initial threshold is established by calculating all the elements' probabilities and selecting a threshold value such that 50% of elements are above it. The upper bound is set such that 10% of elements are above it, while 10% of elements are below the lower bound. The mean cost of solutions produced is used as an approximate measure of the overall quality of the population of solutions. It is recorded after the first iteration and subsequently updated each time the threshold is adjusted. The threshold is adjusted every 20 iterations by examining the proportion of solutions with a cost better than the previously recorded mean. If there are proportionally more solutions with an improved cost, the algorithm is assumed to be in an improving phase and the threshold is raised. If the reverse is the case, the threshold is lowered to allow greater exploration to take place. The new threshold is related to the old by Equation 1.

$$r \leftarrow r + \frac{m_b - m_a}{m} \cdot s \tag{1}$$

$$s = \begin{cases} r_{max} - r & \text{if } m_b - m_a > 0 \\ r - r_{min} & \text{if } m_b - m_a < 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$m$ is the total number of solutions, $m_b$ and $m_a$ are the number of solutions with better and poorer costs than the previous mean respectively, $s$ is a scale factor determined by Equation 2, and $r_{min}$ and $r_{max}$ form a lower and upper bound on the threshold.

**Problem Specific Implementation Details** For the TSP, an ACS element is represented by a city. The cost of an element(city) is simply $d_{ij}$ where $i$ is the previous city and $j$ is the current city.

For the CSP, an element is represented by a car. At each step, each ant adds a new car to its production line schedule. In order to calculate the element/car cost, all of the previous cars must be examined in relation to the new car. If any of these cars are of the same model type as the new car, the separation penalty (i.e. cost) for that model type is recalculated.

### 3.2 Results

The results are given in Table 1. Results are presented as the Relative Percentage Deviation (RPD) from the best known cost, calculated according to $\frac{c - c_{best}}{c_{best}} \times 100$, where $c$ is the cost of the solution and $c_{best}$ is the best-known cost for the corresponding problem. The minimum ("Min"), median ("Med") and maximum ("Max") measures are used to summarise the results as they are non-normally distributed. Given the high consistency of results for CPU time (in seconds), only the median CPU time is reported.

For the TSP, the Elite candidate set strategy appears to offer the best performance in terms of solution cost, while the static candidate set strategy offers the best increase in speed. The Evolving set strategy also produces better solutions than normal ACS in less time, but its solutions are generally more expensive than those produced by Elite candidate set and the static candidate set strategy. It is important to note that the time used by the candidate set strategies is highly dependent on the values of their control parameters.

Further experiments were carried out in which the static set and Elite candidate set strategies were run for equivalent time as the control strategy. These found that their respective performances on cost could be improved if they were run for more iterations, although the static set's performance was still worse than Elite candidate set's.

For the CSP, Evolving set appears to perform best in terms of cost, but is actually slower than normal ACS. Larger candidate set sizes may improve the performance of the static set, which was often found to contain no usable elements requiring all elements to be examined. In contrast, Elite candidate set regularly regenerates its candidate set and does not suffer from this problem. However, Elite candidate set did not perform well on the CSP in terms of solution cost.

## 4 Conclusions

This paper has described some generic candidate set strategies that are suitable for implementation within ACO. This has been a preliminary investigation and it is likely that other candidate set mechanisms, apart from the ones described and tested herein, are possible.

The results indicate that dynamic candidate set strategies can be applied quite succesfully in an ACO setting to combinatorial optimisation problems such as the TSP and CSP. However, each strategy's performance across the two problem types is not consistent. Further investigation into why certain candidate set strategies perform well on some types of problem and poorly on others needs to be carried out. The effects of different control parameter values also need to be more fully explored.

It is conceivable that the results could have been improved by the application of a local search procedure. This has not been done as the primary purpose of this study is to investigate how candidate set strategies can be applied in a general way to constructive meta-heuristics.

In order for ACO meta-heuristics to be used routinely for practical optimisation problems, further empirical analysis of these candidate set techniques needs to be undertaken. Future work will involve studying how the contents of these dynamic candidate sets change with time using these strategies across different problem types.

## References

1. Dorigo, M., Di Caro, G.: The Ant Colony Optimization Meta-heuristic. In Corne, D., Dorigo, M., Glover, F. (eds.): New Ideas in Optimization. McGraw-Hill, London (1999) 11-32
2. Dorigo, M., Gambardella, L.M.: Ant Colonies for the Traveling Salesman Problem. BioSystems **43** (1997) 73-81
3. Stützle, T., Dorigo, M.: ACO Algorithms for the Traveling Salesman Problem. In Miettinen, K., Makela, M., Neittaanmaki, P., Periaux, J. (eds.): Evolutionary Algorithms in Engineering and Computer Science. Wiley (1999)
4. Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications. Springer-Verlag, Berlin (1994)
5. Bullnheimer, B., Hartl, R.F., Strauß, C.: An Improved Ant System Algorithm for the Vehicle Routing Problem. Sixth Viennese workshop on Optimal Control, Dynamic Games, Nonlinear Dynamics and Adaptive Systems, Vienna, Austria (1997)
6. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)

7. Stützle, T., Hoos, H.: Improving the Ant System: A Detailed Report on the MAX-MIN Ant System. Darmstadt University of Technology, Computer Science Department, Intellectics Group., Technical Report AIDA-96-12 - Revised version (1996)
8. Smith, K., Palaniswami, M., Krishnamoorthy, M.: A Hybrid Neural Network Approach to Combinatorial Optimisation. Computers and Operations Research **73** (1996) 501-508
9. Randall, M., Montgomery, J.: Candidate Set Strategies for Ant Colony Optimisation. School of Information Technology, Bond University, Australia, Technical Report TR02-04 (2002)
10. Feo, T.A., Resende, M.G.C.: Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization **6** (1995) 109-133

**Table 1.** Results for each strategy applied to the TSP and CSP

| | | TSP | | | | CSP | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Strategy | Problem | Cost (RPD) | | | CPU | Problem | Cost (RPD) | | | CPU |
| | Instance | Min | Med | Max | Time | Instance | Min | Med | Max | Time |
| Control | gr24 | 0.0 | 0.5 | 5.0 | 17 | n20t1 | 20.7 | 56.0 | 86.2 | 9 |
| | hk48 | 0.0 | 0.4 | 3.4 | 68 | n20t5 | 29.3 | 33.3 | 68.0 | 9 |
| | eil51 | 0.5 | 2.6 | 5.2 | 77 | n40t1 | 42.5 | 58.6 | 73.3 | 31 |
| | st70 | 1.6 | 3.3 | 9.5 | 145 | n40t5 | 27.0 | 32.4 | 47.2 | 31 |
| | eil76 | 1.5 | 3.8 | 8.4 | 170 | n60t1 | 113.8 | 141.8 | 162.5 | 68 |
| | kroA100 | 0.0 | 1.2 | 3.5 | 293 | n60t5 | 24.2 | 33.4 | 50.0 | 67 |
| | d198 | 1.0 | 1.9 | 3.2 | 1144 | n80t1 | 44.2 | 57.3 | 73.0 | 120 |
| | lin318 | 8.8 | 12.3 | 15.9 | 2948 | n80t5 | 24.7 | 34.3 | 47.2 | 119 |
| | pcb442 | 20.1 | 23.9 | 27.7 | 5772 | | | | | |
| | att532 | 20.2 | 26.4 | 31.6 | 8384 | | | | | |
| Static | gr24 | 0.0 | 0.5 | 5.3 | 15 | n20t1 | 5.2 | 14.7 | 25.9 | 15 |
| | hk48 | 0.0 | 0.6 | 2.4 | 30 | n20t5 | 49.3 | 50.7 | 52.0 | 15 |
| | eil51 | 0.5 | 2.6 | 5.2 | 32 | n40t1 | 70.5 | 84.2 | 96.6 | 41 |
| | st70 | 0.3 | 2.0 | 7.9 | 45 | n40t5 | 33.2 | 39.5 | 50.3 | 38 |
| | eil76 | 1.1 | 2.1 | 3.5 | 49 | n60t1 | 196.1 | 209.9 | 273.0 | 79 |
| | kroA100 | 0.2 | 1.1 | 6.4 | 68 | n60t5 | 30.1 | 37.5 | 50.2 | 79 |
| | d198 | 1.6 | 3.0 | 4.7 | 164 | n80t1 | 67.0 | 80.2 | 109.4 | 136 |
| | lin318 | 2.6 | 7.7 | 13.1 | 332 | n80t5 | 40.0 | 53.0 | 63.9 | 138 |
| | pcb442 | 7.0 | 11.0 | 15.6 | 438 | | | | | |
| | att532 | 6.3 | 10.3 | 13.3 | 633 | | | | | |
| Elite | gr24 | 0.0 | 0.5 | 4.4 | 10 | n20t1 | 70.7 | 95.7 | 136.2 | 6 |
| Candidate | hk48 | 0.0 | 0.3 | 2.2 | 38 | n20t5 | 68.0 | 97.3 | 174.7 | 5 |
| Set | eil51 | 0.5 | 3.1 | 6.1 | 43 | n40t1 | 80.8 | 124.0 | 176.0 | 22 |
| | st70 | 0.3 | 2.0 | 9.2 | 83 | n40t5 | 90.9 | 119.3 | 127.0 | 22 |
| | eil76 | 1.3 | 3.4 | 5.9 | 98 | n60t1 | 227.6 | 257.6 | 323.7 | 54 |
| | kroA100 | 0.0 | 0.8 | 5.8 | 168 | n60t5 | 107.8 | 129.2 | 133.6 | 55 |
| | d198 | 0.8 | 1.5 | 2.1 | 686 | n80t1 | 101.2 | 151.4 | 210.6 | 107 |
| | lin318 | 1.9 | 3.5 | 5.1 | 1770 | n80t5 | 92.0 | 123.6 | 154.8 | 109 |
| | pcb442 | 3.1 | 5.8 | 8.4 | 3518 | | | | | |
| | att532 | 3.8 | 4.9 | 6.2 | 5075 | | | | | |
| Evolving | gr24 | 0.0 | 0.6 | 5.0 | 9 | n20t1 | 6.9 | 12.9 | 22.4 | 23 |
| Set | hk48 | 0.0 | 0.1 | 3.4 | 27 | n20t5 | 10.7 | 10.7 | 12.0 | 17 |
| | eil51 | 0.7 | 2.6 | 5.6 | 28 | n40t1 | 3.4 | 13.4 | 21.9 | 86 |
| | st70 | 0.7 | 3.9 | 7.3 | 47 | n40t5 | 5.1 | 5.7 | 10.2 | 72 |
| | eil76 | 1.1 | 2.8 | 6.7 | 62 | n60t1 | 68.4 | 78.0 | 90.1 | 198 |
| | kroA100 | 0.0 | 0.5 | 4.6 | 98 | n60t5 | 2.1 | 2.8 | 4.6 | 174 |
| | d198 | 0.7 | 1.8 | 4.0 | 355 | n80t1 | 10.0 | 15.9 | 22.7 | 358 |
| | lin318 | 3.2 | 5.0 | 6.5 | 954 | n80t5 | 0.5 | 1.8 | 3.9 | 307 |
| | pcb442 | 13.8 | 17.4 | 20.4 | 1861 | | | | | |
| | att532 | 13.5 | 17.1 | 22.8 | 2883 | | | | | |