Candidate set strategies for ant colony optimisation

Randall, Marcus; Montgomery, James

Published in:
Ant Algorithms - 3rd International Workshop, ANTS 2002, Proceedings

DOI:
10.1007/3-540-45724-0_22

Published: 01/01/2002

Document Version:
Peer reviewed version

Licence:
Unspecified

Candidate Set Strategies for Ant Colony Optimisation*

Marcus Randall and James Montgomery†
School of Information Technology
Bond University, QLD 4229

July 17, 2008

Abstract

Ant Colony Optimisation is a maturing class of meta-heuristic search algorithms for discrete optimisation problems that are being increasingly applied to real world problems in areas such as communications and transportation. As these techniques systematically scan the set of possible solution elements before choosing a particular one, 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

Ant Colony Optimisation (ACO) [4] is a relatively new optimisation paradigm encompassing a range of meta-heuristics based on the mechanics of natural ant colonies. These techniques have 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 [3]. The ACO meta-heuristics are population based constructive techniques that allow each agent(ant) to add an element (such as the next city for the TSP) to its solution at each step of the algorithm. The entire set of possible elements is explored in order to determine a suitable element to add to the ant's current solution. Like tabu search [7], the vast majority of an ant algorithm's runtime is devoted to evaluating the neighbours/elements of the problem. As such, 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*. This paper outlines generic 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) [13].

The reader is referred to Dorigo and Gambardella [4] and Dorigo, Di Caro and Gambardella [3] for an overview and background of ACO. This paper is organised as follows. Section 2

*We wish to acknowledge the Australian Research Council for their financial assistance to this research.
†This author is a PhD scholar supported by an Australian Postgraduate Award.

describes how candidate set strategies have been applied to various optimisation problems to date while Section 3 gives a description of some generic candidate set strategies. Section 4 outlines the computational experiments and Section 5 has the concluding remarks.

2 Existing Candidate Set Strategies

This section provides a brief overview of some candidate set strategies that have been applied to both benchmark and real world problems using ACO based techniques. The common theme for these techniques is that they use 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).

2.1 Problems

2.1.1 Travelling Salesman Problem

Stützle and Dorigo [15] have reviewed several ant based techniques for the TSP, including Ant System (AS), Ant Colony System (ACS), $\mathcal{MAX}-\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) and Rank-Based Ant System (AS$_{rank}$). In each of these, ants use pheromone levels and heuristic information in order to choose the next city to add to their solution. Many ACO techniques also apply a local search procedure, such as 2-opt or 3-opt, to improve the best solution found at each iteration. Many studies use candidate sets as part of the local search phase of the algorithm, yet only in later improvements of ACS are candidate sets used within the solution augmentation process [5, 6, 15].

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. Only if the set has been exhausted (i.e. those cities in the set are already in the solution), are the remaining cities considered. This approach has been particularly useful on larger problems (more than 1500 cities) [5, 6]. The nearest neighbour candidate set takes $\Omega(n^2)$ time to compute and it has been shown that maintaining sets of less than 10 cities (less than 1% of the total number of links in one instance) can be sufficient to contain all the links in the optimal solution [12].

The nearest neighbour candidate set can be easily generalised for problems in which each element has a relationship with every other element, such as the TSP. That is, it is possible to determine (or estimate) the cost of adding an element to an ant's existing solution given the last element that was added. The proximity of one element to another is the relative cost of adding that element given that an ant is situated on the other element. However, for problems that do not exhibit strong relationships between elements, such as in the QAP where elements are related to their position in the solution, nearest neighbour type techniques are difficult to apply.

Johnson and McGeoch [8] and Reinelt [12] also discuss candidate sets for the TSP, although they do not consider the application of these to ACO. In addition to the nearest neighbour candidate set, Reinelt proposes a candidate set based on the Delaunay graph. The candidate set produced by the Delaunay graph is augmented by including edges which connect two other edges already in the Delaunay graph. This produces a candidate set with size between $9n$ and $10n$, where n is the number of cities. It must be noted that such an approach can only be used for geometric TSPs and thus is difficult to generalise.

2.1.2 Vehicle Routing Problem

Bullnheimer, Hartl and Strauß [1] have developed an ant system for the vehicle routing problem (VRP) based on their previous work on AS_{rank} [2, 15]. Due to the similarity between the VRP and the TSP, their work extends their work on the TSP. Each ant constructs one or more tours during a single run of the algorithm (starting and ending at the depot) by successively adding customers to its solution. When there are no more customers to be selected (due to vehicle capacity or tour length constraints), the ant returns to the depot and starts a new tour. At each step, customers are selected based on the amount of pheromone deposited between customers as well as a combination of the distance between customers and savings achieved by utilising a vehicle's entire capacity.

For each customer, v, a candidate set is created in which all other customers appear in increasing order of distance from v. As these distances do not change throughout the execution of the algorithm, the candidate sets are created only once in the initialisation phase. Although not explicitly stated, this candidate set strategy is clearly a variant of Elite candidate list as described in Glover and Laguna [7]. The computational experience showed that the use of candidate sets improves both speed and quality of solutions.

2.1.3 Other Problems

Randall and Tonkes [10] have constructed a simple candidate set strategy for the network synthesis problem. For this problem, a network is constructed from a set of possible communication traffic routes. The candidate set is composed of only the routes that are more likely to result in a lower network operational cost.

2.2 Summary of Existing Techniques

While there have been many applications of ACO to different optimisation problems, few of these have made use of candidate set strategies. For the most part, research has focused on the use of candidate sets in local search heuristics so as to improve the results of ant based algorithms. ACS was one of the first to make use of candidate sets, using a nearest neighbour approach for TSP. Bullnheimer et al. [1] also developed a candidate set strategy based on the nearest neighbour heuristic for the VRP. The candidate set approaches suggested by Reinelt [12], such as the Delaunay candidate set, show some promise, although their application may be limited to problems that are geometric.

A common theme with the few instances of candidate set strategies in ACO is to assign values to elements before attempting to solve the problem. Within ant based algorithms, the value of an element need not simply be a measure of its cost, but can be based on its probability as determined by pheromone and cost information. Once the value of elements has been calculated, they can be sorted and the top k elements chosen to form the candidate set. Dynamic candidate set strategies (those in which the set of elements is recalculated periodically throughout the run) are an alternative to static candidate sets. As much of the power of ACO comes from the use of adaptive memory (through the use of pheromone information), it is likely that using dynamic candidate set strategies will lead to further improvements in terms of both solution quality and computational time.

3 Generic Candidate Set Strategies

As is evident from the above, there are two broad ways of calculating candidate sets, a *static* (*a priori*) approach in which the candidate sets are fixed before the commencement of ACO and

dynamic, in which sets need to be calculated and recalculated throughout the search process. The former techniques are more problem specific and suitable for simpler problems such as the TSP.

The tabu search (TS) meta-heuristic first made use of dynamic candidate list strategies [7]. The latter techniques are able to be generalised across different optimisation problem. Hence, their development and refinement is necessary in order to solve complex problems. In this section, the application of candidate set strategies as described in the TS literature [7] will be given in an ACO context. In addition, a new general purpose strategy (called *Evolving Set*) will be described.

In the following, p refers to an element's probability of being added to the solution (see Dorigo and Gambardella [6]) and η is the cost of adding an element to the solution.

1. *Aspiration Plus.* A quality threshold for the elements to be added to an ant's solution is first established. The set of possible elements is searched until an element of this quality (or better) is found. Once this has occurred, the next $Plus$ elements are used to form the candidate set. Quality can be defined in two ways; a) the cost η of the element and/or b) the probability of the element, p.

   To ensure that the set of elements is sufficiently sampled, values of Min and Max are established. Let $first$ be the number of elements examined before the element satisfying the quality threshold is reached. The number of elements examined is given in Figure 1.

Figure 1: Determining the candidate subset using Aspiration Plus.

```
If first + Plus < Min
   Examine Min elements
Else
   If first + Plus > Max
      Examine Max elements
   Else
      Examine first + Plus elements
```

   The threshold value can be varied throughout the search process depending on the quality of the produced solutions. For instance, if the search process is in an improving phase, the threshold value would be high. As an example of establishing thresholds, consider the TSP in which quality is defined by the cost of the edges. Let m be the cost of the smallest edge. Then $\eta_{ij} = \frac{m}{d_{ij}}$ where i and j denote two different cities. The threshold value can now range between 0 and 1 throughout the search process.

2. *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. This set is then used for the next l iterations of the algorithm or until the quality of the elements falls below a critical level. At this time, the candidate set is rebuilt. Again quality can be defined in terms of η or p. The rationale of this approach is that a good element now is likely to be a good element in the future.

   Consider the TSP in the application of the Elite candidate set. In particular, consider a TSP in which groups of cities naturally form clusters[1]. If the elite set is calculated from one of the cities within a cluster, the set would contain those cities (which would be

[1]This approach is also applicable if the problem is only loosely clustered.

relatively close to one another). Once all of the cities within the cluster have been used, the overall quality of the remaining elements would decrease and hence a new set would need to be calculated. This corresponds to an ant jumping between clusters.

3. *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, p, are eliminated temporarily from the search process. These elements are given a "tabu tenure" [7] 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 could be implemented in the same way as the Aspiration Plus strategy.

   Again consider the application of the TSP. Initially, a candidate set is calculated from the start city. Some of these cities will produce low probabilities and hence will be given a tabu status and excluded from the candidate set. The tabu status would sensibly be recorded as a certain number of (ant) steps. After a while, the non-tabu cities (the candidate set) will be exhausted and the status of the tabu cities will change, naturally refreshing the candidate set.

4 Computational Experience

Two problem classes will be used to test the generic candidate set strategies. These are the TSP and the CSP. The CSP is a common problem in the car manufacturing industry [13]. 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 problem instances that we use for both problems are given in Table 1. For those instances that have a proven optimal solution, the optimal cost is displayed, otherwise the best-known cost is shown.

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. The ACS parameter settings are given in Table 2.

For both problem types, a control strategy and an ACS with a static candidate set will be 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 is nearest neighbour (as described in Section 2.1.1). The static set strategy for the CSP is also a nearest neighbour algorithm. For each car, the separation distance to every other car is calculated. The best k form the candidate set for each car. In our experiments, we set $k = 10$ for both TSP and CSP.

4.1 Code Implementation

This section describes the mechanics of implementing the candidate set strategies within the ACS framework. In addition to the three candidate set strategies, a description of how the threshold (used by the Aspiration Plus and Evolving set strategies) is established and varied, as well as problem specific details are given. All programs are coded in the C language.

4.1.1 Aspiration Plus

The implementation of Aspiration Plus follows the high-level description given in Section 3 very closely. Every time an ant chooses its next element, Aspiration Plus generates a candidate set in the way described, beginning its examination of elements from the point where it finished

Table 1: Problem instances used in this study.

Problem Type	Instance	Description	Optimal Cost	Best-Known Cost
TSP	gr24	24 cities	1272	
	hk48	48 cities	11461	
	eil51	51 cities	426	
	st70	70 cities	675	
	eil76	76 cities	538	
	kroA100	100 cities	21282	
	d198	198 cities	15780	
	lin318	318 cities	42029	
	pcb442	442 cities	50778	
	att532	532 cities	27686	
CSP	n20t1	20 cars, class 1		58
	n20t5	20 cars, class 5		150
	n40t1	40 cars, class 1		146
	n40t5	40 cars, class 5		352
	n60t1	60 cars, class 1		152
	n60t5	60 cars, class 5		562
	n80t1	80 cars, class 1		330
	n80t5	80 cars, class 5		772

Table 2: Parameter settings used in this study.

Parameter	Value
β	-2
α	0.1
ρ	0.1
m	10
q_0	0.9
iterations	3000

its examination the last time it was used. To ensure uniformity across problem instances, the values of $Plus$, Min and Max are expressed as a proportion of the number of elements. Using simple parameter tuning, we chose the values $Plus = 0.05$, $Min = 0.15$, and $Max = 0.30$.

4.1.2 Elite Candidate Set

Elite candidate set as described in the TS literature regenerates its candidate set in response to either a predetermined number of iterations or steps passing, or if the quality of elements in the set falls below a critical level. The implementation used in this study only responds to the former, 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.

4.1.3 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 aspiration threshold, which is adjusted by the algorithm (see Section 4.1.4), 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 than the one chosen.

4.1.4 Candidate Set Quality Threshold

A simple heuristic has been developed for adjusting the aspiration threshold periodically between preset upper and lower bounds. This technique equally applies to both the Aspiration Plus and Evolving set candidate strategies. 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 the lower bound is chosen such that 10% of elements are below it. After the first iteration, the mean cost of the solutions produced is calculated and recorded. The mean cost is used as an approximate measure of the overall quality of the population of solutions. 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 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 recorded mean cost is then updated. Equation 1 relates the new threshold to the old.

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

Where:

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.

$$s = \begin{cases} r_{max} - r & \text{if } x_b - x_a > 0 \\ r - r_{min} & \text{if } x_b - x_a < 0 \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

r_{min} and r_{max} form a lower and upper bound on the threshold.

4.1.5 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 of ACS, 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 for that model type is recalculated. This penalty corresponds to η.

4.2 Results

The results are given in Tables 3 through 12 (each table contains the results of a different candidate set strategy applied to either the TSP or CSP). The minimum (**Min**), median (**Med**), maximum (**Max**) and inter-quartile range (**IQR**) measures are used to summarise the results.

As the data are non-normally distributed, non-parametric statistical tests were used to analyse the results. In order to compare the time and cost results across problem instances (but within problem type) the data for CPU time and cost were normalised. All tests for statistical significance were carried out with $\alpha = 0.05$. An analysis of the results for the TSP is presented first.

A Kruskal-Wallace test of the costs achieved by the five strategies indicates that their respective costs are different. Given the large difference between the cost results for Aspiration Plus and the other strategies, this initial test result may be misleading. Hence, a further test was carried out with the data from just the other four strategies, which also reported a significant difference. The Mann-Whitney test was used to make distinctions between pairs of strategies. Compared to all other strategies, the costs achieved by Aspiration Plus are the worst. Compared to the control strategy, Evolving set produces better solutions. The static set strategy also produces better solutions than the control strategy, and has similar performance to Evolving set. Compared to both Evolving set and the static set, Elite candidate set produces better solutions, with both results statistically significant. The costs achieved by Elite candidate set are approximately 1% better on problems with less than 200 cities and 3-6% better on larger problems compared to the static strategy. It also produced the optimal solution for three of the problems, `gr24`, `hk48` and `kroA100` and was within 1-5% of the optimal cost for all other problems except `pcb442`.

There are statistically significant differences in computation time for the five strategies. Comparisons between pairs of strategies showed highly significant differences for all cases. The static set strategy is the fastest. On problems with more than 100 cities, it took less than 15% of the CPU time compared to the control strategy. Relative to the control, its time decreases rapidly as the number of cities grows, as the number of candidates decreases with the inverse

Table 3: Results for control strategy on TSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
gr24	1272	1279	1336	9	17	17	17	0
hk48	11461	11511	11847	85	67	68	68	1
eil51	428	437	448	7	76	77	77	1
st70	686	697	739	25	143	145	146	2
eil76	546	559	583	9	168	170	172	3
kroA100	21292	21543	22030	427	291	293	295	4
d198	15943	16075	16291	188	1132	1144	1151	16
lin318	45716	47190	48723	966	2925	2948	2979	45
pcb442	60970	62898	64858	2010	5703	5772	5838	19
att532	33268	35000	36441	946	8350	8384	8404	34

Table 4: Results for static candidate set on TSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
gr24	1272	1278	1340	5	14	15	15	0
hk48	11461	11527	11732	69	30	30	30	0
eil51	428	437	448	8	32	32	32	0
st70	677	689	728	10	45	45	46	0
eil76	544	550	557	6	49	49	50	0
kroA100	21320	21508	22650	421	67	68	70	0
d198	16029	16255	16514	127	158	164	168	3
lin318	43106	45245	47537	1093	316	332	345	13
pcb442	54341	56371	58693	2210	416	438	465	9
att532	29439	30535	31376	504	588	633	663	8

Table 5: Results for Aspiration Plus on TSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
gr24	1318	1372	1613	87	13	16	17	3
hk48	13972	15369	17819	2609	50	52	64	12
eil51	529	627	654	76	60	62	75	6
st70	924	993	1177	170	107	117	136	5
eil76	710	751	881	52	130	151	160	17
kroA100	34849	36512	42482	6246	236	246	274	22
d198	39431	42552	44316	1808	912	918	929	7
lin318	147488	156621	161806	5708	2440	2441	2443	1
pcb442	180785	190930	197255	3116	4776	4780	4784	2
att532	112958	116829	120926	2937	7001	7017	7077	15

Table 6: Results for Elite candidate set on TSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
gr24	1272	1278	1328	1	10	10	10	0
hk48	11461	11493	11715	100	38	38	38	0
eil51	428	439	452	8	43	43	43	0
st70	677	689	737	12	83	83	83	0
eil76	545	557	570	9	98	98	98	0
kroA100	21282	21460	22522	372	168	168	169	0
d198	15906	16012	16117	63	685	686	687	1
lin318	42832	43501	44188	408	1768	1770	1773	2
pcb442	52372	53714	55068	886	3512	3518	3535	4
att532	28743	29041	29412	331	5072	5075	5089	2

Table 7: Results for Evolving set on TSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
gr24	1272	1279	1336	25	9	9	10	0
hk48	11461	11474	11849	75	27	27	28	0
eil51	429	437	450	12	27	28	29	0
st70	680	702	724	22	46	47	49	1
eil76	544	553	574	11	61	62	64	1
kroA100	21292	21379	22270	240	97	98	98	0
d198	15892	16064	16407	104	350	355	364	2
lin318	43368	44142	44770	648	949	954	1197	8
pcb442	57764	59619	61162	1166	1857	1861	1908	3
att532	31418	32416	34002	1269	2875	2883	3551	13

Table 8: Results for control strategy on CSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
n20t1	70	91	108	16	9	9	9	0
n20t5	194	200	252	16	8	9	9	0
n40t1	208	232	253	15	31	31	32	0
n40t5	447	466	518	21	30	31	31	1
n60t1	325	368	399	20	67	68	68	0
n60t5	698	750	843	31	66	67	67	0
n80t1	476	519	571	29	119	120	121	2
n80t5	963	1037	1136	50	117	119	120	2

Table 9: Results for static candidate set on CSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
n20t1	61	67	73	4	14	15	15	1
n20t5	224	226	228	2	14	15	15	1
n40t1	249	269	287	10	40	41	41	0
n40t5	469	491	529	22	38	38	40	1
n60t1	450	471	567	28	77	79	79	1
n60t5	731	773	844	41	79	79	80	1
n80t1	551	595	691	27	134	136	139	3
n80t5	1081	1182	1265	69	136	138	141	3

Table 10: Results for Aspiration Plus on CSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
n20t1	66	69	71	2	9	9	9	0
n20t5	164	166	168	0	11	11	11	0
n40t1	174	182	185	3	48	49	49	0
n40t5	406	419	440	13	48	51	53	1
n60t1	298	308	317	8	124	125	126	1
n60t5	680	708	737	24	123	129	132	3
n80t1	430	444	454	7	247	250	255	6
n80t5	996	1035	1080	30	252	264	270	8

Table 11: Results for Elite candidate set on CSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
n20t1	99	114	137	11	5	6	6	0
n20t5	252	296	412	18	5	5	6	0
n40t1	264	327	403	22	22	22	22	0
n40t5	672	772	799	37	22	22	23	0
n60t1	498	544	644	39	53	54	55	1
n60t5	1168	1288	1313	96	54	55	56	2
n80t1	664	830	1025	110	106	107	108	1
n80t5	1482	1727	1967	124	108	109	110	1

Table 12: Results for Evolving set on CSP.

Problem	Cost				CPU Time (seconds)			
Instance	Min	Med	Max	IQR	Min	Med	Max	IQR
n20t1	62	66	71	3	23	23	23	0
n20t5	166	166	168	0	16	17	17	0
n40t1	151	166	178	11	86	86	87	0
n40t5	370	372	388	2	71	72	73	1
n60t1	256	271	289	10	197	198	199	1
n60t5	574	578	588	4	172	174	175	1
n80t1	363	383	405	18	356	358	359	1
n80t5	776	786	802	11	304	307	309	2

square of the number of cities. The next fastest strategies were Evolving set and Elite candidate set, that used 34% and 58% the time of the control strategy respectively. Aspiration Plus required 80% of the time used by the control. It is important to note that the time used by the candidate set strategies is highly dependent on the values of their control parameters.

For the CSP, the costs achieved by the five strategies were found to be different. All Mann-Whitney results were significant. In contrast to the TSP results, Elite candidate set performed worst in terms of cost on the CSP instances, often achieving costs twice the best-known. The static set strategy was somewhat better, although its costs on most instances were approximately 50% poorer than the best-known cost. The control strategy performed better than the static set strategy on all instances except for `n20t1`. Aspiration Plus was the next best on cost, while Evolving set achieved significantly better costs than all other strategies.

For the CSP instances, significant differences also exist between the computation time for the five strategies. Evolving set was the slowest, followed by Aspiration Plus, which recorded times between 40% and 70% of those of Evolving set. The static set strategy was faster than Aspiration Plus, and the control strategy was faster than the static strategy. Elite candidate set was the fastest strategy, reporting times of 25-35% those of Evolving set. The static set's slow runtime on the CSP is the opposite of what was expected. Further investigation revealed that the static set often contained no useful elements, requiring the entire set of elements to be examined. In contrast, Elite candidate set regularly regenerates its candidate set and does not suffer from this problem.

Limited experimentation was also conducted with different parameter settings on the TSP. Tests of Aspiration Plus with high values of Min (more than 80% of elements) and Max (100% of elements) resulted in a large improvement in solution cost, although runtimes were more than three times that of the control. Given the speed advantage that the static set strategy and elite candidate list hold over the control, experiments were carried out in which these 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. A fuller investigation into the effects of different parameter settings needs to be carried out to better predict which values are suitable.

5 Conclusions

This paper has described some generic candidate set strategies that are suitable for implementation within ACO. The use of candidate set strategies is necessary in order to ensure the efficient

application of ACO techniques. 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.

For the TSP, the best costs were achieved by Elite candidate set, followed by the static set strategy. Evolving set also performed well on cost compared to the control, although Aspiration Plus performed very poorly. In terms of runtime, the static strategy was best, followed by Evolving set, Elite candidate set and Aspiration Plus, all faster than the control. For the CSP, Evolving set performed the best in terms of cost, attaining costs far closer to the best-known cost than any of the other four strategies. The remaining strategies, in order of cheapest cost to highest, are Aspiration Plus, the control, the static set strategy, and Elite candidate set. It is interesting to note that Aspiration Plus produces better results on the CSP than on the TSP, while Elite candidate set performs far worse on the CSP than on the TSP. The results for runtime are almost exact opposites of those for cost. Elite candidate set was the fastest, while Evolving set was slowest, with the other strategies intermediate between the two. Hence, there is no single "best" strategy across problem type, although Evolving set is consistenty better than the normal ACS control. 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 the strategies' respective control parameters also need to be more fully explored.

In some cases it is conceivable that improved cost results could have be achieved by applying a local search to the best solution found at each iteration. This has not been done here 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. Adding a local search to the algorithm would have confounded the results.

Aspiration Plus's poor performance on the TSP is highly dependent on the values of $Plus$, Min, Max. If these values are too high, the runtime for Aspiration Plus becomes excessive, while its cost performance approaches that of the control strategy. However, setting these values lower reveals a potentially serious flaw in applying Aspiration Plus to constructive meta-heuristics. Aspiration Plus works by sampling a small amount of the surrounding neighbourhood. In iterative meta-heuristics, the neighbourhood is made up of transitions which can, potentially, be reversed. However, with meta-heuristics like ACS, once an element has been added to a solution it remains a part of that solution. Thus, a strategy that limits which elements are considered at each step based more on the total number of elements than on the *value* of individual elements, risks making poor and irreversible choices. Randall [11] describes a variant of ACS for solving dynamic optimisation problems that employs backtracking. It is conceivable that this could also be applied to problems like the TSP to improve the performance of strategies like Aspiration Plus.

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. In general though, more investigation into efficiency issues needs to be carried out. Candidate set strategies are one way to approach this. Another way is to consider parallel implementations of ACO. Some preliminary work has been carried out (for instance) by Stützle [14] and Randall and Lewis [10].

References

[1] Bullnheimer, B., Hartl, R. and Strauß, C. (1997) "An Improved Ant System Algorithm for the Vehicle Routing Problem", Annals of Operations Research, 89, pp. 319-328.

[2] Bullnheimer, B., Hartl, R. and Strauß, C. (1999) "A New Rank Based Version of the Ant System: A Computational Study", Central European Journal for Operations Research and Economics, 7, pp. 25-38.

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

[4] Dorigo, M., Di Caro, G. and Gambardella (1999) "Ant Algorithms for Distributed Discrete Optimization", Artificial Life, 5, pp. 137-172.

[5] Dorigo, M and Gambardella, L. (1997) "Ant Colonies for the Traveling Salesman Problem", Biosystems, 43, pp. 73-81.

[6] Dorigo, M. and Gambardella, L. (1997) "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", IEEE Transactions on Evolutionary Computing, 1, pp. 53-66.

[7] Glover, F. and Laguna, M. (1997) Tabu Search, Kluwer Academic Publishers: Boston.

[8] Johnson, D. and McGeoch, L. (1997) "The Traveling Salesman Problem: A Case Study", In Local Search in Combinatorial Optimization, Aarts, E. and Lenstra, J. (eds), Wiley: Chichester.

[9] Randall, M. and Lewis, A. (2001) "A Parallel Implementation of Ant Colony Optimisation", Journal of Parallel and Distributed Computing (to appear).

[10] Randall, M. and Tonkes, E. (2001) "Solving Network Synthesis Problems using Ant Colony Optimisation", Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence, Monostori, L., Váncza, J. and Ali, M. (eds), 2070, Lecture Notes in Artificial Intelligence, Springer Verlag: Berlin, pp. 1-10.

[11] Randall, M. (2001) "Constructive Meta-heuristics for Dynamic Optimisation Problems", Working Paper.

[12] Reinelt, G. (1994) The Traveling Salesman: Computational Solutions for TSP Applications, 840, Lecture Notes in Computer Science, Springer Verlag: Berlin.

[13] Smith, K., Palaniswami, M. and Krishnamoorthy, M. (1996) "A Hybrid Neural Network Approach to Combinatorial Optimisation", Computers and Operations Research, 73, pp. 501-508.

[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), 1498, Lecture Notes in Computer Science, Springer Verlag: Berlin, pp. 722-741.

[15] Stützle, T. and Dorigo, M. (1999) "ACO Algorithms for the Traveling Salesman Problem", In Evolutionary Algorithms in Engineering and Computer Science, Miettinen, K., Makela, M., Neittaanmaki, P. and Periaux, J. (eds), Wiley: Chichester.