Niching for Ant Colony Optimization

Technical Report: TR006

Daniel Angus

dangus@ict.swin.edu.au

Complex Intelligent Systems Laboratory
Centre for Information Technology Research
Faculty of Information & Communication Technologies
Swinburne University of Technology
Melbourne, Australia

Niching for Ant Colony Optimization

Daniel Angus

Complex Intelligent Systems Laboratory
Centre for Information Technology Research
Faculty of Information & Communication Technologies
Swinburne University of Technology
Melbourne, Australia
dangus@ict.swin.edu.au

Abstract. Ant Colony Optimization (ACO) is a relatively new class of algorithm inspired by the foraging behaviour of biological ants that has shown promise for application to optimization problems. The ability of ACO algorithms to solve more difficult artificial problem instances is an important result for researchers, as these are often more akin to industrial (real-world) applications. While most ACO algorithms are able to find a single (or few) optimal, or near-optimal, solution to difficult (NP-hard) problems, these solutions are often located in the same neighbourhood of solution space. A small change to the problem can have a large impact on a specific solution by decreasing its quality, or worse still, by rendering it infeasible. Over the past 20 years, niching methods, such as fitness sharing and crowding, have been implemented with success in the field of Evolutionary Computation (EC). Such niching methods try to simultaneously locate and maintain multiple optima to increase search robustness - typically in multi-modal function optimization. In this paper it is shown that a niching technique applied to an ACO algorithm permits the niching ACO algorithm to simultaneously locate and maintain multiple areas of interest in the search space, with minimal impact on the quality of solutions found.

1 Introduction

In natural ecologies, a population of organisms is rarely spread uniformly (within an environment), but rather is typically distributed across a wide spatial area and divided into local or sub-groups. Resources available to individuals across a geographical distribution can differ, and so sub-groups of a 'species' may specialise to exploit these differences. The effect of this speciation (or population level natural selection) is referred to as 'niching' in the field of population ecology and population genetics [10, 22].

In a computation sense, niching may permit a more effective use of available resources by a search algorithm by, either implicitly or explicitly dividing and searching different areas of the search space in parallel [17]. Such automatic resource re-allocation is useful in preserving useful population diversity for an extended search time. Niching techniques have also proved particularly useful when problem domain complexity is scaled up to include multiple global and local solutions; problems which are commonly referred to as 'multi-modal'.

This investigation outlines two variants of the ACO algorithm which include forms of niching used in the field of Evolutionary Computation: *fitness sharing* and *crowding*. Fitness sharing involves the

modification of the cost function to create and maintain niches, whereas crowding uses a population-based replacement strategy the effect of which is to modify the explicit historical information maintained by the algorithm. The algorithms presented here are extensions of the population-based ACO algorithm proposed by Guntsch and Middendorf [13, 14]. Their ant-inspired algorithm maintains a history in the form of a population of solutions rather than a pheromone map like other traditional ACO algorithms (although it still utilises a pheromone map for solution construction).

The algorithms presented here are evaluated by their ability to locate and maintain multiple, spatially distributed, near-optimal solutions rather than just the single best solution found overall. The Travelling Salesman Problem (TSP) [20] has received much attention by ACO researchers and is widely understood and accepted as a benchmark combinatorial problem class, as such a simple contrived instance of the TSP has been designed for a brief qualitative analysis. Unfortunately the TSP does not lend itself to quantitative analysis the performance of the algorithms presented. This is due to the difficulty in effectively measuring the modality of standard benchmark TSP instances which is discussed in more detail in Sec. 3.1. Instead, several Continuous Function Optimization (CFO) problems have been chosen as test problems. CFO problems are often used to analyse the performance of niching algorithms [12, 16] due to the ability to easily measure the modality of individual problem instances.

2 Algorithm Definitions

2.1 Population-based ACO

A population-based[1] ACO algorithm called FIFO-Queue ACO is introduced in [14]. This algorithm, although applied to static instances of the TSP and Quadratic Assignment Problem (QAP), is ultimately aimed at addressing dynamic optimization problems. It differs from other ACO algorithms in that it maintains a population of discrete solutions rather than only storing all solution information in a pheromone map. A pheromone map is still maintained, however its contents are always a direct reflection of the current state of the population. As a solution is added (to the population of solutions) its associated components pheromone values are positively updated and as a solution is removed from the population the pheromone values associated with this solution are negatively updated. A more formal description of FIFO-Queue ACO is provided in Alg. 1.

The FIFO-Queue algorithm was extended in [13] with application to dynamic problem instances as well as including additional population maintenance strategies based on age and quality of stored solutions. The FIFO-Queue algorithm was renamed in [13] as population-based ACO (P-ACO) and this has led to some "apparent" confusion amongst researchers who now refer to the original FIFO-Queue algorithm as P-ACO [25, 24, 23]. This is confusing because the authors treat P-ACO as an algorithmic framework (like the ACO meta-heuristic framework [3, 6, 7, 8]), however no "framework" has actually been defined but rather specific implementations. In an attempt to clarify this, a generic framework based on P-ACO is described here and for consistency it is henceforth referred to as the population-based ACO meta-heuristic framework (P-ACO$_F$).

P-ACO$_F$ uses procedures which are previously defined in the ACO meta-heuristic framework: *ants generation & activity*, *pheromone trail evaporation* and *daemon actions*. Readers unfamiliar with

[1] The usage of *population* in this paper is in the EC sense of a collection of stored solutions.

Algorithm 1 FIFO-Queue ACO

```
1: Uniformly initialise pheromone map values to τ0
2: while stopping criterion not met do
3:     Construct solutions
4:     Select best solution
5:     if currentPopulationSize < maxPopulationSize then
6:         Insert best solution into population
7:         Insert best solution information into pheromone map
8:     else
9:         Remove oldest solution information from pheromone map
10:        Remove oldest solution from population
11:        Add new solution to population
12:        Add new solution information into pheromone map
13:    end if
14: end while
```

these procedures should consult [3, 6, 7, 8] for more detailed explanations. P-ACO$_F$ introduces a new procedure called *solution storage & maintenance* which in turn removes the requirement for *pheromone trail evaporation*. A process level description of ACO and P-ACO is provided in Fig. 1.

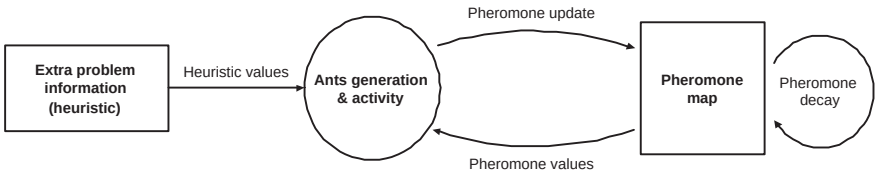
The *solution storage & maintenance* procedure is responsible for the storage of candidate solutions and their translation into pheromone information. Solutions generated by the *ants generation & activity* procedure are added to a finite sized population of solutions. The original FIFO-Queue ACO uses an elite replacement scheme where new solutions are searched and only the best solution is added to the population every algorithmic cycle, replacing the oldest solution. In this work two alternative solution replacement schemes are used.

The way in which the population of solutions interacts with the pheromone map is implementation specific. FIFO-Queue ACO employs an update mechanism in which the pheromone map is persistent and as solutions are added to the population the pheromone map is adjusted to reflect this change. This tight coupling between pheromone map and population results in fast updates to the pheromone map.

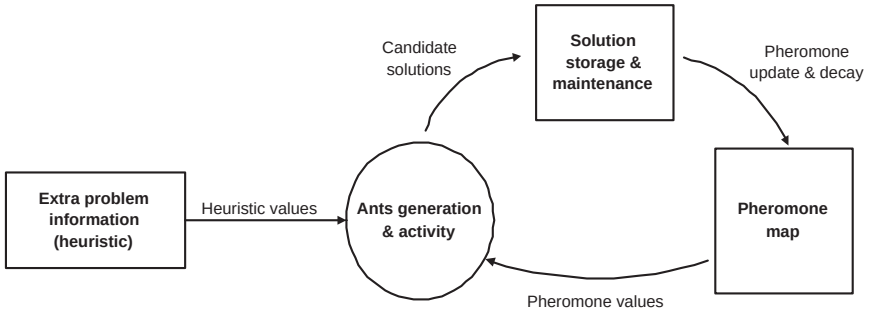
An alternative approach is to use a temporary pheromone map. At every algorithmic cycle the entire current population can adjust the pheromone values of an initially uniform pheromone map which is then used in the creation of the next generation of candidate solutions. After the next generation is created this pheromone map is discarded and a new pheromone map created[2]. In the hardware-based P-ACO$_F$ presented in [23] pheromone values are calculated from the population as required, so that at no point in time does an entire pheromone map exist.

The traditional *pheromone decay* procedure is replaced in P-ACO$_F$ by the *solution storage & maintenance* procedure. At any time the population will contain the most up-to-date solutions obtained by the algorithm and the use of any of the pheromone maintenance schemes outlined in the preceding paragraph eliminates the requirement for decay of specific pheromone values. Solutions are still removed from the population and their associated pheromone information either

[2] The *temporary pheromone map* scheme is used for the implementation of algorithms introduced in this paper, however there is no reason that future implementations could not use the same update scheme as FIFO-Queue ACO.



(a) ACO process organisation



(b) Population-based ACO process organisation

Fig. 1. Original ACO framework (a) versus population-based ACO framework (P-ACO$_F$) (b)

simultaneously removed or not added during the next cycle. The removal of the requirement for pheromone decay was commented on as being an effective computational speed-up in [14].

2.2 Determining the Similarity of Solutions

To implement niching, a metric is required to calculate the similarity of candidate solutions; knowing the spatial relationship of solutions in the search space is fundamental to implementation of a successful niching system. For continuous function optimization a common method used to obtain the difference between solutions, when using a real-value encoding scheme, is to calculate the Euclidean distance (1) where P and S are the solution vectors. For TSP, the difference is obtained by calculating the number of common edges used by two solutions and applying formula (2) to obtain a difference measurement. Using (2), 0 represents the solutions being exactly the same, whereas 1 represents the solutions being completely different, i.e. sharing no common edges.

$$difference_{euc} = \sqrt{\sum_{i=1}^{n} (p_i - s_i)^2}, \text{ where } P = (p_1, \ldots, p_n), S = (s_1, \ldots, s_n) \tag{1}$$

$$difference_{tsp} = 1 - \frac{shared\ edges}{number\ of\ cities} \tag{2}$$

2.3 Fitness Sharing P-ACO

Once calculated, the similarity measurements can be used to derate[3] the quality of a solution according to its proximity to similar solutions. This niching method is known as 'fitness sharing' and was formalised by Goldberg and Richardson [12]. The quality adjustment (Q'_i), niche count (c_i) and sharing functions ($sh\,(d)$) used in this investigation are the same as those outlined in [11] and are reproduced for convenience (3, 4 & 5).

With (3, 4 & 5) if a solution is alone in a neighbourhood (i.e. its nearest neighbour is of a distance $> \sigma_{share}$), then its quality is unaltered since: $sh\,(d) = 1$ for itself, and this will be the only contribution to the niche count (c_i), ($Q'_i = Q_i$ in this case). If two identical solutions are acted upon with the fitness sharing equations then their respective quality will be halved since the niche count (c_i) = 2, and this effectively halves the quality of each solution ($Q'_i = Q_i/2$).

Algorithm 2 outlines the implementation details of Fitness Sharing P-ACO (FS-PACO). In this implementation a temporary pheromone map scheme is used. The *update history* procedure involves all solutions from a single iteration being added to the population and the oldest solutions being removed. This procedure is much like that of the FIFO-Queue ACO algorithm, however it differs in that all solutions are added not just the best-so-far.

$$Q'_i = \frac{Q_i}{c_i} \tag{3}$$

$$c_i = \sum_{j=1}^{m} sh\,(d_{ij}) \tag{4}$$

$$sh\,(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha} & \text{if } d < \sigma_{share}, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Where:

Table 1. Symbols used in 3, 4 & 5

Q_i	: Quality of individual solution i
c_i	: Niche count for solution i
d_{ij}	: Distance (difference) between solution i and solution j
$sh\,(x)$	: Sharing function
σ_i	: Niche radius
α	: Power used to modify the shape of the sharing function

[3] *Derate* is a common term in niching literature and refers to a solution's quality being negatively adjusted.

2.4 Simple Crowding P-ACO

The EC *crowding factor* model was first formalised as a diversity maintenance scheme by DeJong [5], and later reworked as a niching strategy by Mahfoud [16]. Labelled *deterministic crowding*, it was designed to slow convergence and maintain diversity by limiting dominant building blocks in a population. The replacement policy in deterministic crowding involves a candidate solution competing (based on solution quality) for a place in the history with the most similar solution taken from a subset of the total population of current solutions. If a candidate solution is better than its most similar rival, the rival is replaced, otherwise the candidate solution is discarded. Deterministic crowding is useful when we are selecting a subset of the population to produce the next generation of solutions, for example, the parent candidate solutions in a Genetic Algorithm (GA). Since we are not selecting parents, as in a GA, another form of crowding has been implemented: *simple crowding* [2]. Simple crowding is a specialisation of Restricted Tournament Selection [15].

Simple crowding involves each candidate solution being compared against the entire current population of solutions. The closest current member of the population is replaced with the candidate solution if the candidate solution is of better quality. This replacement strategy eliminates duplicate solutions in the history, which has the effect of widening the focus of the search. Algorithm 3 outlines the Simple Crowding P-ACO algorithm (SC-PACO).

2.5 Implementations of P-ACO$_F$ for TSP and CFO

For function optimization problems the pheromone representation used by all algorithms tested (FIFO-Queue ACO, SC-PACO and FS-PACO) is that which was defined by Socha [24]. The method described in this paper is to maintain a population of solutions where each stored solution correlates to a Probability Density Function (PDF) in the pheromone map. The pheromone map is defined by the bounds of the test function. The height (and width) of each PDF is governed by the corresponding solution's quality and age (age in relation to the time it has persisted in the population). By adding multiple PDFs into an initially uniform pheromone map a probability landscape is created which can be sampled. For FS-PACO and SC-PACO implementations tested in this paper, solution age is not used in the calculation of PDFs so the height and width of PDFs are based on the solution quality alone. Solution quality is obtained (for the minimisation case) by applying the adjustment (6) to the value obtained from the CFO problem addressed. The minimum (f_{min}) and maximum (f_{max}) function values used in (6) are continually updated during algorithm execution which eliminates the requirement to know the function bounds a-priori.

For the TSP the pheromone map is uniformly initialised and as is standard with most ACO algorithms each pheromone value corresponds to an edge connection of the search graph. In FS-PACO and SC-PACO a greedy pheromone contribution strategy is used (7) where the parameter λ is used to control the greedy bias of solution quality contribution to the pheromone map.

$$solution\ quality = 1 - \frac{function\ value - f_{min}}{f_{max} - f_{min}} \tag{6}$$

$$\Delta\tau = \left(\frac{path\ length_{best}}{path\ length} \right)^{\lambda} \tag{7}$$

Algorithm 2 Fitness Sharing P-ACO: FS-PACO

```
1: while stopping criterion not met do
2:     Construct temporary pheromone map
3:     Construct Solutions
4:     Update history
5:     De-rate quality
6: end while
7: procedure De-rate quality
8:     for j = 1 to numberOfAnts do
9:         nicheCount ← 0
10:        for k = 1 to historySize do
11:            if difference < σ then
12:                shareValue ← (1 − (difference/σ)^α)
13:            else
14:                shareValue ← 0
15:            end if
16:            nicheCount ← nicheCount + shareCount
17:        end for
18:        history[j].quality ← history[j].quality/nicheCount
19:    end for
20: end procedure
```

Algorithm 3 Simple Crowding P-ACO: SC-PACO

```
1: while stopping criterion not met do
2:     Construct temporary pheromone map
3:     Construct solutions
4:     Simple crowding history update
5: end while
6: procedure Simple crowding history update
7:     for j = 1 to numberOfAnts do
8:         for k = 1 to historySize do
9:             if difference < leastDifference then
10:                leastDifference ← difference
11:                closestHistoryIndex ← k
12:            end if
13:        end for
14:        if history[closestHistoryIndex].quality > ant[j].quality then
15:            history[closestHistoryIndex] ← ant[j]
16:        end if
17:    end for
18: end procedure
```

3 Results

3.1 Travelling Salesman Problem (TSP)

Although it is often easy to visualize a TSP from a tour creation perspective, it is much more difficult to visualize the solution space of a TSP than that of a CFO. Techniques exist to map all solutions to an n-dimensional TSP into a 2-dimensional space [27], however these techniques fail to preserve the neighbourhood relationship of the TSP. This lack of visualization technique contributes to the difficulty in accurately measuring the modality of standard benchmark TSP instances, and therefore the normal criterion used to evaluate the performance of the niching algorithms (peak location and maintenance) is unavailable.

Instead of using a quantitative analysis method a qualitative analysis is used to show that on a trivially simple TSP instance a state-of-the-art ACO algorithm (Max-Min Ant Systems [9, 26]) fails to locate and maintain both of the two distinct optima past convergence[4], whereas FS-PACO and SC-PACO locate and maintain both optimal locations past convergence. The TSP instance is called the 'crown' problem due to the shape of the two optimal solutions . It is a symmetric, 2-Dimensional Euclidean TSP containing 6 vertices (Fig. 2), is bi-modal (two global optima) with the optima being a reasonable distance apart (difference = 0.5) [5]. The bi-modality of the crown problem means that a conventional ACO algorithm is likely to only locate and maintain a single optimum, whereas an alternate approach such as niching should locate and maintain both optima.

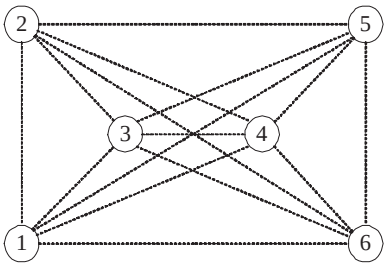
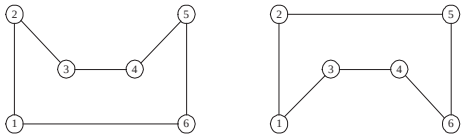


Fig. 2. Crown TSP problem visual representation



(a) A - 1,2,3,4,5,6 length = 541.42

(b) B - 1,2,5,6,4,3 length = 541.42

Fig. 3. Optimal solutions to crown problem

The two optimal solutions present in the crown problem are shown in Fig. 3. For comparison purposes the MMAS algorithm was tested[6], and the number of times either of the two optima were found was recorded (Fig. 4). This experiment was repeated 100 times for consistency of reported

[4] MMAS was run without a restart strategy as it was intended to show that in a *single run* of the MMAS algorithm only one optimal solution is obtained. The MMAS algorithm if employed with restart would no doubt be able to locate both optima on this trivially simple TSP, however, the point being made here is that there is no mechanism in MMAS to stop the algorithm from continually re-finding the same optimal solution.

[5] The coordinates of this dataset are: {{0,0},{0,100},{50,50},{100,50},{150,100},{150,0}}

[6] Using the standard parameters as in [26]

results. The graphs presented in Fig. 4, Fig. 5 and Fig. 6, although illustrating single experimental runs, are indicative of each algorithms behaviour[7].

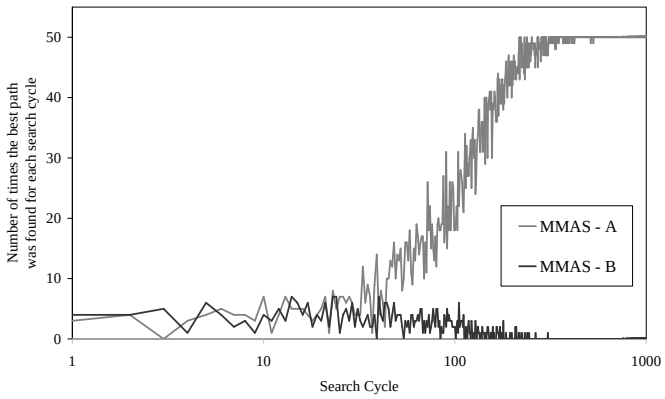


Fig. 4. Crown problem: Occurrence of the two optimal paths over time using MMAS (A & B are the two distinct optimal paths)

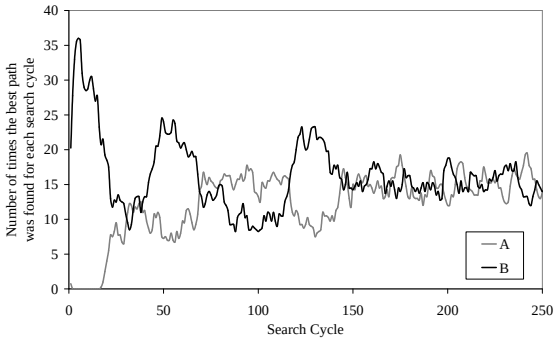


Fig. 5. Crown problem: Occurrence of the two optimal paths over time using FS-PACO (A & B are the two distinct optimal paths)

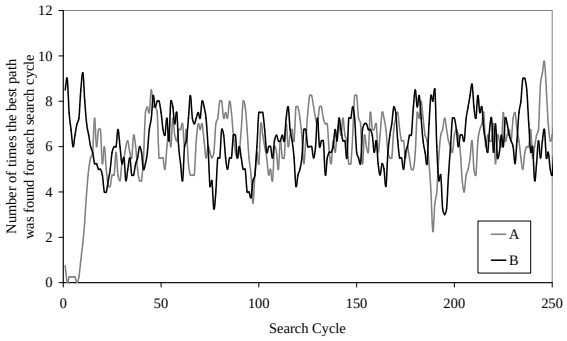


Fig. 6. Crown problem: Occurrence of the two optimal paths over time using SC-PACO (A & B are the two distinct optimal paths)

Due to the small search space size of the problem it was not surprising that MMAS was able to find both optimal solutions in a short time, however this algorithm, which at the start was able to find both optima, quickly converged on a single optimal solution. FS-PACO exhibited satisfactory optima location and maintenance past convergence. Unlike MMAS, FS-PACO was able to locate and maintain both optimal solutions for the duration of the search. SC-PACO exhibited good optima location and maintenance characteristics as the optimal solutions (when found) were quickly added

[7] Parameters used in all experimentation are located in the Appendix.

into the population where they remained past convergence of the algorithm due to the replacement strategy employed.

The crown problem has demonstrated that on a simple problem, fitness sharing and simple crowding PACO are able to locate and maintain multiple optimal solutions in a single run, whereas under the prescribed configuration MMAS does not. This analysis was extended to a larger (albeit still small) TSP instance, the berlin52 problem [21]. The same observations of peak location and maintenance held for this problem as did on the small crown problem. A typical graph of performance of a single experimental run using the FS-PACO algorithm is included here to highlight that as the dimensionality of the problem increases that niching effects can still be readily observed (Fig. 7). It must be pointed out that this figure does not suggest that this niching effect increases the algorithms ability to obtain the single 'global' optima, even though in this case the global optima was found.

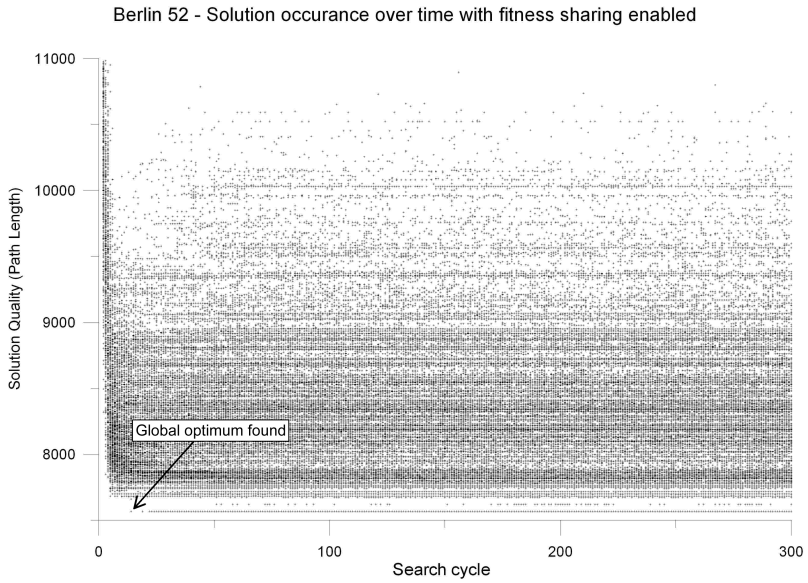


Fig. 7. Berlin52 - Occurrence of solutions over time using FS-PACO (Where each point represents a solution). Note the horizontal lines that indicate the location & maintenance of near-optimal solutions typical of niching algorithms.

As it is not possible to easily measure the modality of standard TSPs a quantitative assessment of these algorithms niching ability on these problems cannot be produced. It is for this reason that attention has been concentrated on CFO, a class of problem whose modality is readily available.

3.2 Continuous Function Optimization (CFO)

This study is concerned with each algorithm's ability to obtain not just a single best solution but multiple good solutions. The performance criterion used in this study is the number of distinct

optima located and maintained. An optimum is considered found if a solution exists that is within a specified radius of this optimum. Within this radius the solution is considered to be close enough to the optimum that if desired a local-search technique would be able to find the exact location of the optimum. For all function tested the radius was set to 0.1% of the bounds of the function. The stopping criteria for all algorithms was set to 50,000 function evaluations which is well past the convergence point for all algorithms tested for completeness and consistency.

Four common test functions (restricted in 2-Dimensions for easy visualization) were selected to be tested and are listed in Tab. 2. FS-PACO and SC-PACO were tested on these functions and their performance is compared to FIFO-Queue ACO.

$$f(n) = \sqrt{\sqrt{\left(\sum_{i=1}^{n}((i+8)+0.1)\right)^2} + \sqrt{\left(\sum_{i=1}^{n}((i+2)+0.2)\right)^2} + \sqrt{\left(\sum_{i=1}^{n}(i-8)\right)^2}} \tag{8}$$

$$f(x,y) = a\left(y - b\times x^2 + c\times x - d\right)^2 + e\left(1-f\right)\times cos\left(x\right) + e \tag{9}$$
$$\text{Where: } a = 1, b = 5.1/\left(4\pi^2\right), c = 5/\pi, d = 6, e = 10, f = 1/\left(8\pi\right)$$

$$f(x,y) = \frac{\left(\left(x^2+y-11.0\right)^2 + \left(x+y^2-7.0\right)^2\right)}{2186} \tag{10}$$

$$\frac{1}{f(x,y)} = \frac{1}{500} + \sum_{j=1}^{25}\left(\frac{1}{j + \sum_{i=1}^{2}\left(\left(x-a_{ij}\right)^6 + \left(y-a_{ij}\right)^6\right)}\right) \tag{11}$$
$$\text{Where: } a_{ij} = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & \cdots & 32 & 32 & 32 \end{bmatrix}$$

Table 2. Multi-modal test functions

Function name	Equation	Reference	Number of distinct optima	Bounds
3 Pot-holes	(8)	[19]	3	$-10 \le x \le 10, -10 \le y \le 10$
Branin's R-Cos	(9)	[1]	3	$-5 \le x \le 10, 0 \le y \le 15$
Himmelblau's	(10)	[4]	4	$-6 \le x \le 6, -6 \le y \le 6$
Shekel's Foxholes	(11)	[5]	25	$-65.54 \le x \le 65.54, -65.54 \le y \le 65.54$

The results (Tab. 3) are encouraging for SC-PACO which for two of the test instances was able to locate and maintain all distinct optima. Furthermore, SC-ACO located and preserved all three optima in 94% of trials on the 3 pot-holes function and located and preserved all three optima in 85% of trials on the Branin's R-Cos function. The results were not as encouraging for the FS-PACO algorithm which demonstrated less than ideal performance on all four test functions. On average,

Table 3. Results of testing FS-PACO & SC-PACO on a suite of 2-Dimensional multi-modal CFO problem instances compared to the FIFO-Queue ACO algorithm (results averaged over 100 trials)

Algorithm	Problem	Number of optima	Optima located (Average)	Variance
FS-PACO	3 Pot-holes Function	3	40.33%	5.66%
	Branin's R-Cos Function	3	54.66%	7.66%
	Himmelblau's Function	4	54.75%	17.00%
	Shekel's Foxholes	25	41.80%	13.68%
SC-PACO	3 Pot-holes Function	3	98.00%	1.66%
	Branin's R-Cos Function	3	95.00%	4.33%
	Himmelblau's Function	4	100%	0%
	Shekel's Foxholes	25	100%	0%
FIFO-Queue ACO	3 Pot-holes Function	3	33.33%	0%
	Branin's R-Cos Function	3	33.33%	0%
	Himmelblau's Function	4	25.00%	0%
	Shekel's Foxholes	25	4.00%	0%

FS-PACO was able to locate and preserve close to half of the distinct optima for the test functions used. It is hypothesised that the reason for some of this poor performance was due to the sensitivity of the niche radius parameter (σ_i). Effective configuration of this parameter is dependent on apriori knowledge of the spacing and size of optima in the search space [28].

The performance of FIFO-Queue ACO was the poorest, as expected. This algorithm was not designed for niching and as such was only able to identify (although to its credit consistently) a single optima for all four functions tested. Selection of the best solution per iteration for insertion into the population results in this algorithm quickly converging on an individual optimum. We speculate that the SC-PACO algorithm applied to higher dimension CFO problem would still exhibit useful niche formation and maintenance, however as the dimensionality of the problem instance increases so too should the history size[8].

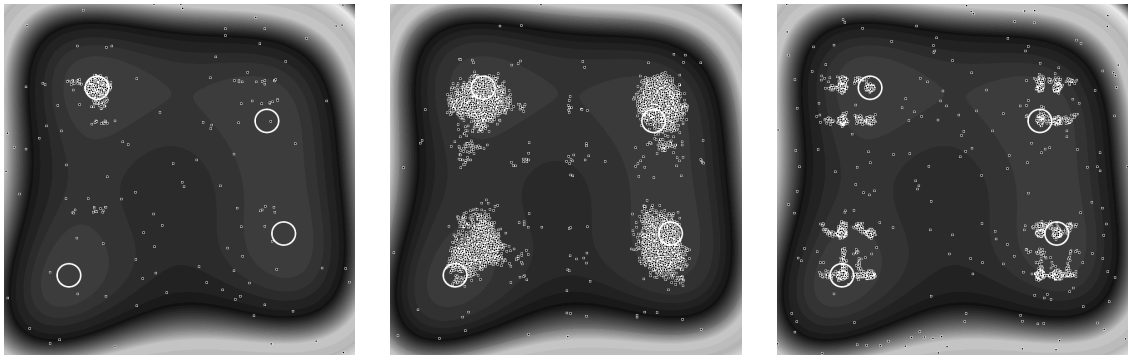
Three figures showing all sampled points for an entire experimental run (using FIFO-Queue ACO, FS-PACO & SC-PACO) on Himmelblau's Function are included (Fig. 8). Figure 8(a) clearly shows the convergence of FIFO-Queue ACO to a single optimum. Figure 8(b) and Fig. 8(c) highlight the FS-PACO and SC-PACO algorithms ability to locate and maintain multiple optima [9].

4 Conclusions and Future Work

Both niching methods evaluated in this study were shown to be effective at identification and maintanence of multiple optimal solutions past convergence for an illustrative TSP instance and several CFO instances of varying modality. For all problem instances none of the non-niching ACO algorithms tested were able to maintain multiple optimal solutions. The niching methods presented assist in the maintenance of useful diversity. This diversity is expected to improve algorithm robustness when applying ACO algorithms to more complex multi-modal problem domains such as

[8] It was observed in preliminary experimentation that if the modality of the problem is larger than the history storage SC-PACO fails to maintain all useful niches.

[9] This software is available from `http://www.it.swin.edu.au/personal/dangus/`



(a) FIFO-Queue ACO (b) FS-PACO (c) SC-PACO

Fig. 8. All samples generated in a single run by algorithms applied to Himmelblau's Function (White circles indicate the four distinct optima, small black & white square represent samples)

dynamic problems. In this type of problem dynamic changes can render some existing solutions infeasible, and thus storing a wider range of solutions lessens the chance of the algorithm history becoming completely infeasible.

The effects of niche radius (σ_i) and other parameters are not discussed in this paper as the aim of this paper is to demonstrate a proof of concept of niching ACO. A larger study into the effects that these parameters have on the algorithms behaviour on a wider range of test problems is underway. It is also clear that the niching methods implemented in this investigation represent only a subset of available niching techniques, however this paper has demonstrated that niching with ACO algorithms is feasible and effective.

Other interesting areas for future work will be the implementation of niching-like behaviour in multiple colony ACO algorithms as it may allow for more effective information sharing between colonies. There is evidence to support the hypothesis that simple multi-colony schemes, which only share best-so-far information, are somewhat ineffective [18]. In this recent paper the authors suggest that a niching-like information sharing mechanism would help independent colonies avoid searching the same areas of search-space. We speculate that a niching extension applied to multi-colony ACO would allow for an increase in the diversity of solutions (thereby increasing robustness) without adversely affecting quality of the solutions since each independent colony can be allowed to concentrate on a single (distinct) area of the search space.

References

[1] F. K. Branin. A widely convergent method for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research and Development*, pages 504–522, September 1972.

[2] J. Brownlee. Parallel niching genetic algorithms: A crowding perspective. Master's thesis, Swinburne University of Technology, 2004.

[3] O. Cordon, F. Herrera, and T. Stützle. A review of the ant colony optimization metaheuristic: Basis, models and new trends. *Mathware & Soft Computing*, 9(2,3), 2002.

[4] K. Deb. Genetic algorithms in multimodal function optimization. Master's thesis, Department of Engineering Mechanics, University of Alabama, Tuscaloosa, AL, 1989.

[5] K. A. DeJong. *An analysis of the behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[6] M. Dorigo, E. Bonabeau, and G. Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16:851–871, 2000.

[7] M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)*, pages 1470–1477. IEEE, 1999.

[8] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. Technical report, IRIDIA, 2000.

[9] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, London, 2004.

[10] N. Eldredge. *Macroevolutionary Dynamics: Species, Niches and Adaptive Peaks*. McGraw Hill, 1989.

[11] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, 1989.

[12] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[13] M. Guntsch and M. Middendorf. Applying population based ACO to dynamic optimization problems. In *ANTS '02: Proceedings of the Third International Workshop on Ant Algorithms*, pages 111–122, London, UK, 2002. Springer-Verlag.

[14] M. Guntsch and M. Middendorf. A population based approach for ACO. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl, editors, *EvoWorkshops*, pages 72–81. Springer-Verlag, 2002.

[15] G. R. Harik. Finding multimodal solutions using restricted tournament selection. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 24–31, San Francisco, CA, 1995. Morgan Kaufmann.

[16] S. W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois, 1995.

[17] S. W. Mahfoud. *Evolutionary Computation 2: Advanced Algorithms and Operators*, chapter Niching Methods, pages 87–92. Institute of Physics Publishing, UK, 2000.

[18] M. Manfrin, M. Birattari, T. Stützle, and M. Dorigo. Parallel ant colony optimization for the traveling salesman problem. Technical Report 2006-007, IRIDIA, March 2006.

[19] B. Prime and T. Hendtlass. Evolutionary computation using island populations in time. In B. Orchard, C. Yang, and M. Ali, editors, *Innovations in Applied Artificial Intelligence: IEA/AIE 2004*, volume 3029 of *LNCS*, pages 573–582. Springer, May 2004.

[20] G. Reinelt. TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.

[21] G. Reinelt. Tsplib95, 1995. Available at: `http://www.iwr.uni-heidelberg.de/groups/comopt/software/tsplib95`.

[22] R. E. Ricklefs. *Ecology*. Thomas Nelson & Sons Ltd, 1973.

[23] B. Scheuermann, K. So, M. Guntsch, M. Middendorf, O. Diessel, H. ElGindy, and H. Schmeck. FPGA implementation of population-based ant colony optimization. *Applied Soft Computing*, 4(3):303–322, August 2004.

[24] K. Socha. ACO for Continuous and Mixed-Variable Optimization. In M. Dorigo, M. Birattari, and C. Blum, editors, *Proceedings of ANTS 2004 - Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *LNCS*, pages 25–36. Springer-Verlag, Berlin, Germany, 5-8 September 2004.

[25] K. Socha and M. Dorigo. Ant colony optimization for continuous domains. Technical Report 2005-037, IRIDIA, December 2005.

[26] T. Stützle and H. Hoos. Improvements on the ant system, introducing the MAX-MIN ant system. In *ICANNGA97 - Third International Conference on Artificial Neural Networks and Genetic Algorithms*, University of East Anglia, Norwich, UK, 1997.

[27] S. Tsubakitani. A two-dimensional mapping for the traveling salesman problem. *Computers & Mathematics*, 26:65–73, 1993.

[28] R. K. Ursem. When sharing fails. In *Proceedings of the Third Congress on Evolutionary Computation (CEC-2001)*, pages 873–879, 2001.

Acknowledgement

The author acknowledges the invaluable assistance received by Jason Brownlee and also acknowledges the guidance Prof. Tim Hendtlass has provided as well as the rest of the team at CISCP.

Appendix

Table 4. Parameters used for testing Crown problem

Algorithm	Search Cycles	Population size	Number of Ants	Pheromone Power	Heuristic Power	Alpha	Sigma	Decay	Lambda
MMAS	500	-	50	2	-1	-	-	0.02	-
FS-PACO	800	20	50	10	-1	1	0.2	-	2
SC-PACO	800	2	50	1	-2	-	-	-	2

Table 5. Parameters used for function optimization problems

Algorithm	Population size	Number of Ants	Initial random sample	Pheromone Power	Alpha	Sigma
FIFO-Queue ACO	50	25	-	2	-	-
FS-PACO	50	25	-	5	2	2
SC-PACO	50	25	50	2	-	-

