Control and Cybernetics

vol. **26** (1997) No. 3

Cooperative distributed search: the ants' way

by

Mark Wodrich* and George Bilchev"""*

* Communication Research Group, University of Cape Town, South Africa,
E-mail: mwodric@eleceng.uct.ac.za

** AA&T, BT Laboratories, Ipswich IP5 7RE, UK
E-mail: george.bilchev@bt-sys.bt.co.uk

Abstract: This paper describes a novel evolutionary optimisation algorithm based on an ant colony metaphor. The algorithm utilises positive feedback to encourage local search in areas where improvement continues to be made, resulting in autocatalytic convergence of search agents to promising regions. The algorith is tested on several standard unconstrained and constrained optimisation problems, and the results are compared with those achieved by existing evolutionary algorithms.

Keywords: evolutionary computation, stochastic optimisation, hybrid genetic algorithms, ant colony optimisation

1. Introduction

This paper investigates a form of evolutionary algorithm based on an ant colony search metaphor. Originally developed for combinatorial optimization problems, the ant colony metaphor has recently been applied to continuous function optimization. The algorithm utilises positive feedback to encourage local search in areas where improvement continues to be made, resulting in autocatalytic convergence of local search agents to promising regions.

Coupled with this local search capability, the algorithm utilizes genetic-algorithm inspired operations to exchange information between high-fitness solutions, resulting in effective global search of the problem space and improved convergence to the global maximum. The continuous ant colony algorithm can thus be seen as a form of hybrid genetic algorithm with local search. The novelty in our approach is in the autocatalytic interaction among the many distributed local search agents which controls the well known exploration/exploitation tradeoff.

Further, the algorithm utilises a simple yet powerful method for handling constraints to achieve excellent performance on constrained problems, making it a promising choice for problems where identifying the feasible region is the major difficulty.

The paper is organized as follows. Section 2 outlines some previous research results on the ant colony optimization method. Section 3 describes the algorithm in detail. Section 4 presents experimental results for both constrained and unconstrained problems as well as comparison with other known evolutionary search algorithms. Conclusions are given in Section 5.

2. Previous research on ant colony optimization

The Ant Colony optimization algorithm for continuous functions is based in part on the natural behaviour of ant colonies, and grew out of the work done by Dorigo, Maniezzo and Colomi (1991; 1996), on combinatorial optimisation. Before the continuous optimization algorithm is explained in detail, it is important to examine the natural basis for the algorithms, and the way this has been applied to combinatorial problems.

2.1. Ant colony behaviour

One of the problems studied by ethologists is to understand how almost blind animals like ants could manage to establish the shortest route between their nest and a food source. It was found that the most important medium of communication among individuals regarding which path to follow consists of pheromone trails. A moving ant lays pheromone (in varying quantities) on the ground as it moves, thus marking its journey by a trail of this substance.

While an isolated ant moves essentially at random, an ant encountering a previously laid trail can detect it, and is likely to follow it, thus reinforcing the trail with its own pheromone. The collective behaviour that emerges is a form of autocatalytic, or positive feedback, behaviour. The more ants that follow a trail, the more attractive it becomes to other ants. Thus the probability of an ant following a trail is proportional to the number of ants that previously followed that trail.

How does this enable ants to find the shortest route around an obstacle? Simulations can be created that simulate a landscape with "cyber-ants" moving around it depositing pheromone, and vividly show the emergent behaviour that causes ants to reinforce the shortest route available.

The SimulAnt program, Wodrich (1996), was written to demonstrate the collective behaviour of ants foraging for food. In the simulation ants arc able to sense pheromone deposited on the landscape by other ants, and their movement is affected by the presence of strong pheromone trails. No "artificial intelligence" is possessed by the individual ants, and the emergent behaviour is purely as a result of the pheromone mechanism. Sample displays from the simulation are

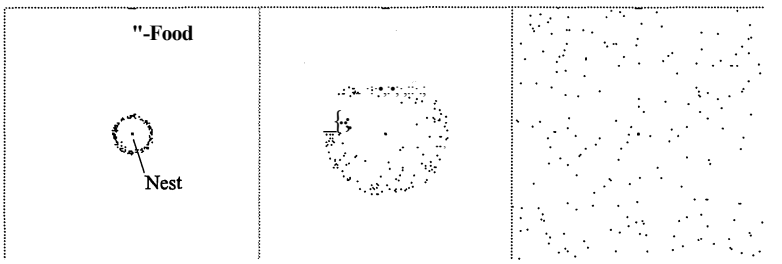

Figure 1. Ants emerging from nest, spreading across landscape Initially no pheromone trails exist to guide the ants towards food. They pour from the nest in random directions, quickly covering the entire landscape.

given in figures one to three. Of course, the actual behaviour of ant colonies is far more complex. The underlying principle of co-operation by positive feedback, however, is seen to be a powerful mechanism. Since it allows ants to effectively minimise the distance they travel, their behaviour offers inspiration for solving optimisation problems where the shortest route must be calculated. In the next section an ant-colony inspired algorithm for such problems is described.

2.2. Combinatorial optimisation

There is a class of optimisation problems that is ideally suited to be solved by ant colony techniques. For example, the famous Travelling Salesman Problem (TSP) is an extremely difficult problem at large scales (i.e., NP-hard), and many different techniques have been developed to solve it. The TSP is defined as follows: "Given a set of N cities with the distance between each pair of cities, find the shortest tour that includes each city exactly once, and ends with the city of origin." Each city can be represented by a number from 1 to N and a distance matrix d is created such that dij gives the distance from city i to city j. A tour can then be represented by a vector of N numbers, and the distance travelled is easily computed. A recent algorithm inspired by ant colonies, called the Ant System, is described below.

2.2.1. The ant system algorithm

The Ant System (AS), Dorigo, Maniezzo and Colorni (1996), is described as applied to the Travelling Salesman Problem. While some of the features of the algorithm are tailored to the TSP, it has been shown that the same approach can be applied to other combinatorial problems, Dorigo, Maniezzo and Colorni (1996).

Given a set of N towns, the matrix d is created to contain the distance between towns (as previously described). There are a constant number of m

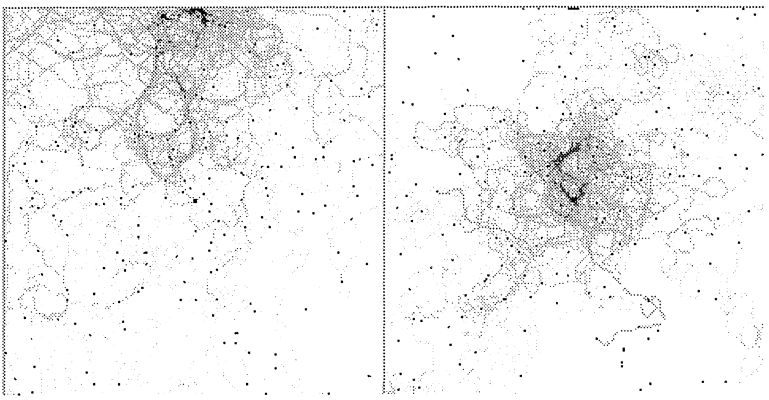

Figure 2. Pheromone trails from food and nest. Soon, ants find food and start depositing pheromone as they try to return to the nest. Since no trails exists to lead them back to the nest, they move essentially randomly. Eventually an ant will find the nest by chance, and start depositing pheromone trails as it re-emerges seeking food.

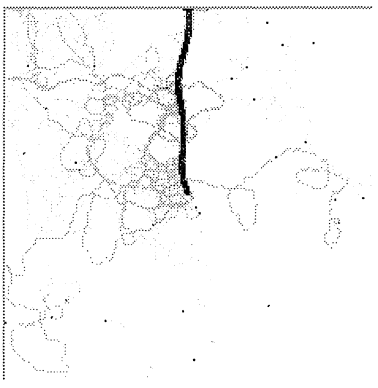

Figure 3. Shortest route to food. Quickly the ants converge to a single route - the shortest path from nest to food.

ants dispersed among the cities, so that at discrete time t there are bi (t) ants in town i. Initially ants may be dispersed randomly among the cities, or all in the same city. (Performance is slightly improved if random dispersion is used). Ants move to another city at the start of each time unit, and select which city to move to based on the pheromone trail and distance between cities. Initially the trail quantity is set to a small value for all routes (to cause ants to select cities with equal probability).

Each ant is a simple agent that:

- Chooses a town to move to with a probability that is a function of the distance and the amount of pheromone on the connecting route.
- Only makes legal tours (it can only visit each city once). To implement this, each ant has its own tabu list, which stores the cities an ant has already visited. An ant can then only move to a city if it is not in the list, after which the city is added to the list.
- After completing a tour of the cities (every N time units, if there are N cities), the ant's tabu list is cleared, and pheromone is laid on the routes travelled during the tour.

If τ_{ij} (t) is the amount of pheromone trail at time t on the route linking cities i and j, then the equation for updating the trail is given by:

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$$

where p E [O, 1] is a constant governing the rate of pheromone evaporation:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k},$$

where m is the number of ants, and

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if ant k uses the route from city i to } j \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

where Q is a constant, and Lk is the tour length of the k-th ant.

Since the amount of trail added is inversely proportional to the distance travelled by an ant, it is clear that shorter routes are more attractive. The positive-feedback mechanism described previously will cause the ants to quickly start following similar routes, until they all converge towards the shortest route found. However, it is possible to give the ants some "intelligence" to further accelerate the process, by enabling them to factor the distance between town into their decision-making process.

The probability of an ant selecting a valid city (not in the tabu list) is given by:

$$p_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha}[1/d_{ij}]^{\beta}}{\sum_{k \notin \text{Tabu List}}[\tau_{ik}(t)]^{\alpha}[1/d_{ik}]^{\beta}}$$

1. Initialise pheromone trail on all routes.
2. Place ants at random locations and clear tabu lists.
3. For $t = 1$ to N select which city each ant moves to and add it to tabu list.
4. Compute length of each ant's tour and preserve the shortest tour found so far.
5. Evaporate pheromone.
6. For $ant = 1$ to M add pheromone trail to routes used by ant.
7. Repeat from step 2 until maximum number of iterations or ants stagnate

Figure 4. The ant system algorithm for the TSP

A summary of the algorithm is presented in Figure 4. The algorithm is remarkably simple, although there are slightly more complex variations (mainly altering the way pheromone trail is added). For a more detailed discussion of the Ant System and results obtained on several standard problems, the reader is referred to Dorigo, Maniezzo and Colorni, (1991; 1996).

The above decribed algorithm cannot be directly applied to engineering design problems where there are both discrete and continuous variables. An algorithm suitable for such problems, based on ant-colony behaviour and incorporating a similar selecting mechanism to the Ant System, has been developed initially by Bilchev and Parmee (1995; 1996). The next section describes this algorithm.

3. Algorithm description

The algorithm is based almost entirely on the algorithm described by Bilchev and Parmee (1995; 1996). However, some modifications have been made that improve performance. Aspects not fully covered in the previous papers are described in detail, as they are vital to effective global function optimisation. Since the algorithm was originally developed for fine local search after promising regions have been identified using other techniques, the previous papers concentrated almost entirely on the local search process. The algorithm described in this paper includes a global search mechanism, resulting in a stand-alone optimisation algorithm.

While the Ant System performs well on combinatorial problems, it cannot be directly applied to continuous optimisation problems. Combinatorial problems always have a finite number of possible solutions (e.g., N factorial), while this is not true for continuous problems. One cannot directly represent such problems as connected graphs (cities linked by routes, etc.), and other representations are needed.

What characterises the co-operation mechanism of the Ant System, is that ants select where to go from a finite set of destinations, with the selection probability based on pheromone quantity. In order to implement this in continuous

problem space, it is necessary to create a finite number of destinations in the problem space (called regions).

While each region acts as a destination for the local ants to explore, it can also be seen as a trial solution to the optimisation problem. The encoding of region positions depends largely on the chosen local and global search techniques. Since the techniques described in this paper use mathematical operations to modify positions, it is more natural to use base 10 encoding; region positions are encoded as vectors of real numbers.

These regions are initially distributed randomly in the problem space, and evolve over time due to the actions of the search agents (ants), so that they gradually become located in regions of high fitness. The regions are updated as the result of two different search processes: local search and global search. Since there are two search processes, the search agents (ants) must be divided between them. The ratio of agents performing local search (local agents) to agents performing global search (global agents) can be varied, and the division of agents is one of the parameters of the ACO algorithm. Generally it is efficient to split the agents 20%-80% between local and global search.

As mentioned previously, global search is important for avoiding local maxima, and can be generally seen as a process that produces large fluctuations in a trial solution. Given the ant-colony metaphor, and the presence of a finite number of regions as described previously, some mechanism must be provided to allow ants to search large areas surrounding the region centre. Termed "random walk" and "trail diffusion" in the original papers, Bilchev and Parmee (1995; 1996), this process is implemented using aspects of Genetic Algorithms to allow regions to exchange information.

It is important to realise that the actual local and global search methods employed are not fixed by the algorithm It is possible to integrate any local or global search method into the ACO algorithm, making it possible to increase performance by using problem-specific methods. While more complex methods are likely to increase performance of the algorithm, this is not guaranteed. This paper focuses on a combination of deliberately simple search methods, to illustrate the power of the ant colony metaphor. The local search method described is a variation of Stochastic Hill-climbing, while the global search technique borrows extensively from Genetic Algorithms. While the GA-like method performs well, it has no biological link to ant-colony behaviour. The link between the algorithm and the ant-colony metaphor is therefore in the recruitment mechanism.

3.1. Local search

The local search technique described is based on Stochastic Hill-climbing, but utilises the pheromone trail leading to regions to influence the search process. Each region is also given an age that stores the number of unsuccessful local search attempts, used to scale the size of the region explored. The basic outline

of the search process is as follows:

1. An ant selects a region with a probability proportional to the pheromone value of the "route" to that region. (The ant is imagined to follow a path from a virtual nest to the region's position.)
2. Once at the region, the ant moves a short distance and the fitness at this point is calculated.
3. The direction the ant moves is either:
   - The same as the direction climbed by the previous ant that selected this region, if that ant found an improved fitness value.
   - A random direction otherwise.
4. If the ant obtains a higher fitness value, the region is moved to the ant's position, the ant deposits pheromone proportional to the improvement made in the region's fitness, and the region's age is decremented.
5. Otherwise, the age of the region is incremented.

The probability of a region being selected by an ant is given by:

$$Pi(t) = \frac{Ti(t)}{\sum_{k} Tk(t)}$$

where i is the region index and Ti(k) is the pheromone trail on region i at time t. Initially the trail value for all regions is initialised to a constant value T_o, so that regions are selected with equal probability. This value grows due to trail addition in step 4 above, where trail is added in proportion to the increase in fitness. The reverse process of evaporation causes attention to be diverted from regions that fail to provide improved fitness. This is analogous to food-source exhausting, where ants rapidly loose interest in a route to food once all the food has been transported back to the nest. Unlike real ants, however, it is possible for the ACO to mistakenly re-search a region, since there is no mechanism to record previously exhausted regions. (This would be difficult to achieve in large search spaces.) Thus, it is possible for a region to "migrate", through local search, to a previously exhausted local maximurri.

The size of the step made by an ant depends on the age of the region, with the size decreasing with increasing age. This enables ants to adapt the area of the local search, improving convergence if a higher fitness is obtained successively in the same direction.

The step size varies linearly between a Maximum Search Radius (for an age of zero), and a Minimum Search Radius (for an age equal to the Maximum Age, a constant parameter). These radii arc fractions of the available variable ranges to search, enabling the local search to scale automatically for small or large variable ranges.

3.2. Global search

As previously mentioned, the global search procedure consists of two GA-inspired methods, random walk and trail diffusion.

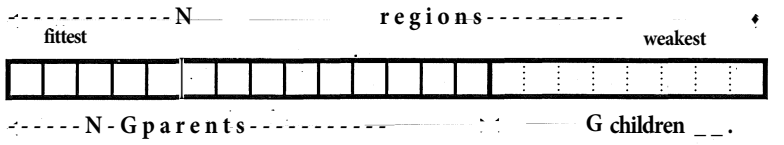

Figure 5. Creation of new regions

The global search process creates G new regions (where G is the number of global search agents). These new regions replace the weakest existing regions, implying that the total number of regions must be greater than the number of global search agents. The new regions are created by selecting information from the remaining (fitter) regions, using GA operations such as crossover and mutation. (See Figure 5). The new regions can be seen as "children" of the remaining "parent" regions.

Random walk can be seen as the creation of new regions by ants exploring the entire problem space. As a real-world analogy this may be seen as foraging members of the ant colony that attempt to discover new food sources. Once a possible food source has been discovered, these ants must attract the remaining ants by depositing a pheromone trail. Within the ACO framework, the new region must be given a pheromone value to attract local agents. The region can then be verified as a "food source" through the local search process.

While there are many possible implementations of the random walk process, the one chosen employs both stochastic steps in region positions and genetically motivated "inheritance" of position information. The process consist of:

1. Creating a new region based on information fl.,Ill parent regions (analogous to crossover in Genetic Algorithms).
2. Adding a random step to the "child" region (analogous to mutation in Genetic Algorithms), with a probability equal to a defined Mutation Probability.

To control the selection of parents for the new region, a Crossover Probability is used. Initially a single parent is chosen, and the first element of the child's position vector is set to the corresponding clement in the parent. For each subsequent element, a new parent may be selected with a probability equal to the above Crossover Probability (CP). Thus, if the CP is equal to 1.0, each element in the child's position has a different parent. For a CP of 0.0, each element has the same parent (and the child is thus identical. to the parent). The need for a tunable CP is born out by empirical results as will be demonstrated in Section 4.

In addition to knowing when to select a parent, a method of selection is also needed. In the ACO, all parents have equal probability of being selected.

The "mutation" process is responsible for ensuring that the entire search space is covered by the random walk process, by adding a random step to

the child's position. Since the need for global search decreases as the search process continues, it is desirable to scale the step size during the run of the search process. This enables the algorithm to focus the search in ever decreasing regions around known optima. This, in turn, increases the probability of locating maxima, since the density of the search increases during the run of the algorithm. Each element of the child's position vector has a fixed probability (Mutation Probability) of being mutated by this operation, otherwise it is left unaltered.

The scaling of the global search is achieved by non-linearising the mutation step size. At the start of the search process, the entire problem space is equally likely to be searched. As the search progresses, the probability of large steps decreases sharply. The step size is obtained from the following equation:

$$\Delta(T, R) = R \cdot (1 - r^{(1-T)^b})$$

where r is a random number from [0, 1], R is the maximum step size, T is the search iteration [0, 1], and b is a positive parameter controlling the degree of non-linearity. The step is applied to each element of the position vector, with the polarity of the step equally likely to be positive or negative. The value R is then the maximum step size that will allow the element to remain within its defined range. (The value 7.1 with the range [0, 10] will have an R of 2.9 [+step] and 7.1 [-step].) The value of T is obtained by dividing the current iteration number by the maximum number of iterations. Thus at the start of the search process T will be 0, and it will increase linearly to 1 at the end of the search process.

Once the "child" region has been created by following these two steps, the fitness value is evaluated. It is then necessary to set the pheromone trail value to encourage local search of the new region. Since the trail value for a region is a function of the local search performance, it is difficult to reliably set the trail value for a newly created region with no local search having been performed on it. If the assigned trail value is too low relative to the other regions, it is unlikely the new region will be chosen for local search. However, if the assigned trail value is too large, the region will unfairly attract local agents from other, possibly more promising regions.

A heuristic rule is that the assigned trail value should be between the lowest and highest trail value of the parent regions. Using the average trail value of the parents means that the probability of the new region being selected for local search is equal to the average probability of a parent being selected. In this way, the new region is likely to be selected, but is not likely to unfairly attract local agents.

The second global search method employed, Trail Diffusion; is similar in many respects to arithmetic crossover in Genetic Algorithms. The position of a child region produced by trail diffusion is obtained from the weighted average of a number of parent regions. Whereas in the random walk process, one parent is selected at a time, trail diffusion selects a pair of parents. The Crossover

1. Initialisation:
2. R = number of regions, L = number of local agents, G = number of global agents
3. Iterations = 0
4. Position regions at random locations, evaluate fitness, and set trail to zero.
5. Iterations = Iterations + 1
6. Send ants:
7. Send G Global agents, which do not deposit pheromone, and update global maximum.
8. Send L Local agents, which may deposit pheromone, and update global maximum.
9. Evaporate trail for all regions.
10. Repeat from Step 5, until maximum number of iterations.
11. Display global maximum.

Figure 6. The ACO algorithm

Probability (CP) is again used to indicate when a new pair of parents is to be selected, and parents are again selected with equal probability.

There are three possible values for each element of the child's position vector:

1. The corresponding- element from parent 1.
2. The corresponding element from parent 2.
3. A weighted average of the above two:

   xi(child) = (a.) · xi(parent1) + (1 - a.) · xi(parent2)

   where a. is a uniform random number from [0, 1]

The probability of option 3 above being used is equal to the Mutation Probability, with options 1 and 2 equally probable otherwise (i.e., if the Mutation Probability is 0.5, the probability of options 1 and 2 being selected is 0.25 each). The fitness of the new region is calculated, and the trail value set to the average trail of the parent regions (as for random walk).

3.3. Pheromone evaporation

An important aspect of the ACO algorithm is the pheromone communication algorithm. To implement a process similar to food-source exhausting in real ants, it is necessary to decrease the amount of pheromone globally, so that without the addition of trail, a region will no longer be selected by ants. (This feature is a form of "forgetting", which is used in many algorithms to improve diversity and avoid premature convergence).

The evaporation process is described by the equation Ti(t + 1) = p • Ti(t), where p is the Evaporation Rate, and Ti(t) is the trail associated with region i at time t

To summarise, an outline of the algorithm is given in Fig. 6.

3.4. Constrained optimisation

The algorithm described above has no features that enable it to effectively handle constrained optimisation problems. However, with very few modifications, it is possible to adapt the algorithm to effectively handle constraints.

The primary feature incorporated into the algorithm is the concept of constraint violation. Given a set of constraints applying to a problem, every trial solution can be tested to determine whether it lies within the feasible region. The constraint violation is a measure of how far outside the feasible region a solution lies, and is calculated as the sum of the violations of each constraint. If the problem contains equality constraints, the feasible region may have infinitely small area. The constraint handling mechanism described below is unable to handle this case. One can either eliminate a variable (if the equality is an analytic equation), or change it into an inequality constraint:

$$G_i(x) = 0 \text{ becomes } |G_i(x)| \leq E, \text{ where } E \text{ is the desired accuracy.}$$

For example, given the constraints: $x + y \leq 10$ and $x \cdot y \geq 1$ (for a 2-dimensional problem), the solution (1, 2) has a constraint violation of 0. The solution (11, 0) would have a constraint violation of 2.

During the local search process ants search for local improvements in fitness. The improvement in fitness acts like a food source that is exploited by the ants. With the constraint handling mechanism, a point is only accepted as a "food source" if its constraint violation is below an acceptable threshold. The acceptable constraint violation changes over time, decreasing linearly from an initial value to the desired final constraint tolerance. This causes ants to be drawn back into the feasible region as the search progresses, but allows limited exploration within the non-feasible region.

The final adjustment needed to the ACO algorithm is a method of ranking regions in order or fitness, so that newly created regions replace the weakest existing regions. (See Figure 5). For unconstrained problems regions are ranked based on the objective function value only. Constrained problems make use of the constraint violation to further discriminate between regions.

A region R1 is considered fitter than another region R2 iff:

- The two constraint violations are equal (or below the acceptable threshold), and the objective function value of R1 is greater than that of R2, or
- The constraint violation of R1 is less than that of R2.

3.5. Discrete ACO (DACO)

The above algorithm is naturally applicable to many real-world problems. However, comparisons between base 10 encoded, and base 2 encoded (discrete) algorithms, are unfair. To enable fair comparison of the ACO algorithm with exisiting algorithms such as Population-Based Incremental Learning (PBIL),

Baluja (1994), and binary-coded GAs, a discrete version of the ACO algorithm is needed.

Instead of re-formatting the algorithm to operate with bit-string representations of a solution, it is far simpler to adapt the real-coded algorithm. The approach taken is as follows: All position vector elements arc forced to have discrete integer values (0.0, 1.0, 2.0, etc.) The algorithm using discrete values only is called Discrete ACO (DACO), whereas the original ACO algorithm using real-coding is called Continuous ACO (CACO).

The DACO algorithm differs in three respects from the CACO algorithm:

1. The initial regions must be created with discrete values.
2. The local search process must only take integer steps.
3. The global search processes must maintain the discrete nature of the elements by altering the random walk (mutation) operation and the trail diffusion (arithmetic crossover) operation, to produce discrete valued children.

Alteration 1 is easily achieved by using a discrete random number generator (or by rounding off random floating-point numbers).

Alteration 2 is achieved by slightly changing the local search process. The region age is no longer needed to scale the local step size; the step size is always 1. The polarity of the step for each variable is defined by a vector where the sign of the elements indicates whether the corresponding variable is to be incremented or decremented. Thus the position vector (1, 2, 3) and the climb direction [+1, -1, -1] produce a new position vector (2, 1, 2). As in the original ACO, a climb direction is preserved for future local agents to use if it leads to increased fitness. Unsuccessful climb attempts cause the direction vector to be randomised.

Alteration 3 involves two changes: The random walk process must produce discrete values, so the non-linearity of the search scaling must be expressed in a form that produces discrete steps in variables. The equation for the step size is now:

$$\delta(T, R) = \lceil R \cdot (1 - r^{(1 - T)^b}) \rceil$$

where r is a random number from [0, 1], R is the maximum step size, and b is a parameter controlling the degree of non-linearity (as before). The function $\lceil x \rceil$ returns the smallest integer greater or equal to x (it is called "ceil", for ceiling, in most computer maths libraries).

The trail diffusion process must be altered so that the arithmetic crossover process returns a discrete value:

$$xi(child) = \lceil a \cdot xi(parenti) + (1 - a) \cdot :r,i(parent2) \rceil$$

where a is a uniform random number from [0, 1]

These simple modifications allow the algorithm to operate in discrete problem spaces, and makes comparison with other discrete-coded algorithms possible. For example, to test the DACO on a function and compare the results with

those obtained with a GA using 9 bits per variable, with each variable having the range $-2.56 \leq x_i < 2.56$:

1. The discrete range of each variable is set to [0, 511] since there are 512 discrete values represented by 9 bits.
2. The continuous range of each variable is set to [-2.56, 2.56), enabling the DACO to automatically convert from discrete to real numbers. The objective function code does not necessarily need to be altered when one switches from CACO to DACO, since the discrete to real conversion can be done transparently before the function is evaluated.

The DACO does not rely on a particular coding scheme (Binary or Gray), since the algorithm does not operate on individual bits or groups of bits. Since the only operator used is addition, the underlying representation of the value is not important. To implement the algorithm one can use either integer (Binary-coded bit string) or floating-point data types.

Since some engineering problems contain mixed types of variables (some continuous, some discrete), the integration of DACO with CACO could prove extremely useful. This would allow fine exploration of continuous variables while still producing valid discrete values where needed. For example, finding an optimal microwave filter design would involve choosing from the available set of component values, with the length of transmission line sections being continuously variable.

4. Experimental results

This section describes the various test functions used to evaluate the performance of the ACO algorithm. All functions are maximisation problems. (Where necessary, the original minimisation problems are transformed to maximisation problems by negation.) Comparisons are made between the final results obtained using the ACO and various other algorithms (depending on the results available in the literature).

The ACO algorithm is tested using (at most) the same number of objective function evaluations. Where discrete coding has been used by other algorithms, the DACO algorithm is used with the same precision, to enable fair comparison. To illustrate the difference in performance between DACO and CACO for these problems, the results obtained using CACO are also given.

Plots of the ACO algorithm's performance on most of the test functions are given in the Appendix.

For all test functions the value given for DACO and CACO is the average maximum value found, averaged over 10 independent runs. The same ACO parameters are used for all test functions (unless otherwise specified):

- 100 Ants, comprising 20 local agents and 80 global agents. Of the global agents, 70 perform the "random walk" operation and the remaining 10 perform "trail diffusion".
- 200 Regions.

Method	Average Maximum
DACO	3905.93
Hill-Climbing	3905.93
PBIL	3905.93
GA	3905.93
CACO	3905.93

Table 1. Results for De Jong F2

- The number of iterations is chosen so that the number of function evaluations is less than or equal to the evaluations needed by the other algorithms compared.
- Evaporation Rate = 0.9
- Mutation Probability = 0.5
- Crossover Probability = 1
- Initial trail value = 1
- Non-linearity of random walk: b = 10
- Maximum region age = 20

Given A ants, R regions, and I iterations, the number of function evaluations is equal to $R + A \cdot I$. The regions are evaluated once at the start of the run, and new regions arc evaluated by the global agents at each iteration.

4.1. Unconstrained problems

4.1.1. De Jong

De Jong (1975), test functions are commonly used to test GA's. Both are normally minimisation problems, and have been converted to maximisation problems Baluja (1994). The standard accuracy used by GAs and the Population-Based Incremental Learning (PBIL) is 12 bits, and thus the discrete range used by the DACO algorithm is [0, 4095] for each problem.

4.1.2. De Jong F2 (2D)

The function is:

$$(3905.93) - 100(x^2 - y)^2 - (1 - x^2), \quad -2.048 \leq x, y \leq 2.048$$

By inspection the global maximum of 3905.93 is seen to occur at $(x, y) = (1, 1)$. The results obtained, with 6000 evaluations for each method, are shown below.

Method	Average Maximum
DACO	55.0
PBIL	55.0
GA	53.5
CACO	55.0

Table 2. Results for De Jong F3

4.1.3. De Jong F3 (5D)

The function is:

$$f(x) = 25.0 - \sum_{i=1}^{5} \lfloor x_i \rfloor, \quad -5.12 \leq x_i \leq 5.12$$

where $\lfloor x \rfloor$ is the smallest integer less than or equal to x.

The function has a maximum value of 55.0 for all $-5.12 \leq x_i < 5.0$. Due to the "flooring" of the variables, the function is characterised by flat areas linked by abrupt steps.

The results obtained, with 6000 evaluations for each method, are shown below:

4.1.4. Griewangk (10D)

The problem was originally a minimisation problem, and was converted to a maximisation problem in Baluja (1994). Restricted to 2 dimensions the surface is seen to be highly multimodal, with many like-sized peaks near the origin. Results given in Baluja (1994) indicate that PBIL and GAs have great difficulty in finding the maximum value of 10 located at the origin.

The function is:

$$f(x) = \frac{2^{1}}{0.1 + \left(\sum_{i=1}^{10} \frac{x_i^2}{405} - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right)}, \quad -512 \leq x_1, \ldots, x_{10} \leq 512$$

The global maximum of 10 occurs at the origin.

Again, an accuracy of 12 bits is used for comparing the performance of DACO to the results published by Baluja using GA and PBIL. The results obtained are:

4.1.5. Homaifar

These two problems are used in Fogel (1995). Fogel uses evolutionary progTamming, and compares his results with Homaifar's results using a GA. Originally

Method	Evaluations	Average Maximum
DACO	50,000	10.0
Hill-Climbing	200,000	10.0*
PBIL	200,000	5.49
GA	200,000	3.52
CACO	50,000	10.0

(* Baluja reports obtaining 10.6223, which cannot be correct since the global maximum is 10.0)

Table 3. Results for Griewangk

the problems were minimisation problems, and have been converted to maximisation problems for use by the ACO by negating the original objective functions.

Although the problems are constrained optimisation problems, they are transformed into unconstrained problems using a penalty function to penalise non-feasible solutions. To enable comparison with the other methods, the ACO uses the same technique. However, since the objective functions changed to maximisation problems, the penalty term is subtracted from the objective function value (instead of being added).

Details of the penalty functions are not included here for brevity, and can be found in Fogel's paper. Since the method used by Fogel, Evolutionary Programming (EP), relies on real-coded solutions, the CACO algorithm is applied to this pair of test function, and the results compared with those obtained using EP. The GA results given in Fogel's papers are included for comparison, although the coding method used (bit-string or real) and accuracy were not specified. For further comparison, the DACO algorithm is applied as well, using 12-bit accuracy.

Function 1 (2D) The objective function is

$$f(x_1, x_2) = -((x_1 - 2)^2 - (x_2 - 1)^2)$$

Subject to the constraints:

$$X_1 - 2X_2 + 1 = 0, \quad -4^{\pi} - X_2^2 + 1 \geq 0$$

with the variable ranges being:

$$-1.82 \leq X_1 \leq 0.82, \quad -0.41 \leq X_2 \leq 0.92$$

The number of ants used by the DACO is 40, to correspond to the population size used by the Evolutionary Programming (EP) algorithm. The results obtained are:

Method	Evaluations	Average Maximum
CACO	4,000	-1.3773
EP	4,000	-1.3789
GA	40,000	-1.4339
DACO	4,000	-1.4609

(The EP and GA results were converted from minima to maxima by negation, since the objective function was negated for use by the ACO algorithm).

Table 4. Results for Homaifar F1

Method	Evaluations	Average Maximum
CACO	4,000	31019.049
EP	4,000	31006.267
GA	40,000	30175.804
DACO	4,000	30963.941

(The EP and GA results were converted from minima to maxima by negation, since the objective function was negated for use by the ACO algorithm).

Table 5. Results for Homaifar F2

Function 2 (5D) The objective function is:

$$f(x1, \ldots, x5) = 5.3578547x + 0.8356891x1X5 + 37.293239x1 - 40792.141$$

subject to the constraints:

$$0 \text{ S } 85.334407 + 0.0056858x2x3 + 0.00026x1x4 - 0.0022053x3X5 \text{ S } 92$$

$$90 \text{ S } 80.51249 + 0.0071317x2,1:5 + 0.0029955.1:1.1:2 + 0.0021813x \text{ S } 100$$

$$20 \text{ S } 9.300961 + 0.0047026x3X5 + 0.0012547x1x3 + 0.0019085x3x4 \text{ S } 25$$

with the variable ranges:

$$78 \text{ S } X1 \text{ S } 102, \quad 33 \text{ S } X2 \text{ S } 45, \quad 27 \text{ S } X3, \ldots, X5 \text{ S } 45$$

The number of ants used by the DACO is again 40, to correspond to the population size used by the Evolutionary Programming (EP) algorithm. The results obtained are:

Method	Average Maximum (xlO0)
DACO(CP= 0.0)	6.84
PBIL (Gray code)	2.62
DACO(CP= 0.5)	2.45
PBIL (Binary code)	2.12
SGA (Binary code)	1.96
DACO(CP= 1.0)	1.77
Hill-Climbing	1.21
CACO (CP= 0.0)	5.44
CACO (CP= 1.0)	1.26

Table 6. Results for Baluja F I

4.1.6. Baluja {lOOD)

These three problems are to be found in Baluja (1995). All the problems use bit-string encoding with 9-bit precision. The DACO algorithm is used for comparison with a discrete range of [0, 511].

For all three functions the variable ranges are $-2.56 \leq x_i \leq 2.56$, $i = 1, \ldots, 100$. A small constant $C = 1e^{-5}$ is added to the denominator of the functions to avoid division-by-zero.

Function 1

$$Y1 = x1, \quad Yi = xi + Yi\text{-}1, \quad i = 2, \ldots, 100$$

$$ti(x) = \frac{1}{C + \left| Y1 I + I{:}i \quad IYil \right|}$$

The global maximum of 100,000 occurs at the origin.

There is a high degree of inter-dependence between elements of a trial solution (small changes in early portions of the trial solution have a cascade effect on subsequent values of Yi). For this reason, the crossover operation is unlikely to create regions of high fitness: trial solution elements cannot be mixed between different regions. Thus, better results are obtained using a Crossover Probability (CP) of 0.0, as opposed to the default of 1.0. Results using both values of CP are given below.

The results obtained, with 200,000 evaluations for each method, are shown in Table 6.

Function 2

$$Y1 = X1, \quad Yi = Xi + \sin(Yi\text{-}1), \quad i = 2, \ldots, 100$$

Method	Average Maximum (xlO0)
DACO(CP=0.0)	12.79
DACO(CP=0.5)	8.80
DACO (CP=1.0)	8.28
PBIL (Gray code)	5.61
GA-Scale (Gray code)	4.63
PBIL (Binary code)	4.40
Hill-Climbing (Gray code)	4.38
CACO (CP=0.0)	8.77
CACO (CP=1.0)	5.52

Table 7. Results for Baluja F2

$$h(x) = \frac{1}{C + j\text{IYII} + \text{L}i=1^{100} \text{IYil}\, j}$$

Again, the global maximum of 100,000 occurs at the origin.

As in function 1, there is a high degree of inter-dependence between elements of a trial solution. Again, the results obtained using a CP of 0.0 and 1.0 are shown.

The results obtained, with 200,000 evaluations for each method are shown in Table 7.

Function 3

$$h(x) = \frac{1}{C + \text{L}i=1^{100}\, \text{j}0.024(i + 1) - \text{Xii}}$$

The continuous global maximum of 100,000 occurs at $xi = 0.024(i + 1)$

Using 9-bit accuracy, the global maximum is 416.64

Since there is no inter-dependence in this function (the summation term for a particular is independent of the value of other x/s), the default Crossover Probability of 1.0 is used.

The results obtained, with 200,000 evaluations for each method, are shown in Table 8.

4.1.7. Artificial neural network weight optimisation

Recently evolutionary algorithms have been applied to evolving weights of artificial neural networks (ANNs) Hart (1994), Michalewicz (1995). The test function involves finding the optimal weight values for the 46 trainable weights in a simple parity network described in Baluja (1995).

Method	Average Maximum (xlO0)
Hill-Climbing (Gray code)	416.64
PBIL (Gray code)	366.77
DACO	253.80
GA-Scale (Gray code)	210.37
PBIL (Binary code)	16.43
GA-Scale (Binary code)	12.30
Hill-Climbing (Binary code)	8.10
CACO	54231

Table 8. Results for Baluja F3

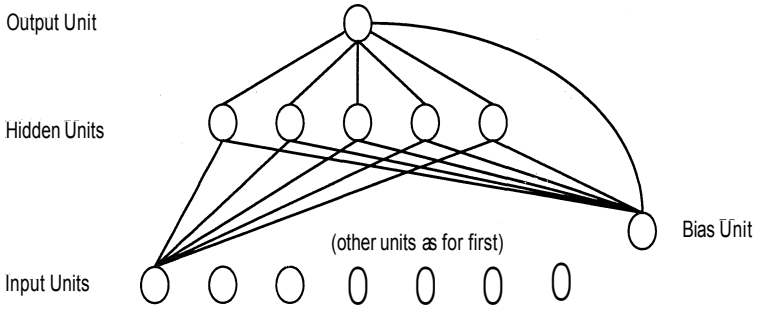

Figure 7. ANN Topology

The object of the ANN is to calculate the parity of 7 input bits. The bits are either 0 (represented by -0.5) or 1 (represented by +0.5). If the parity was 1, the target output of the ANN is +0.5, and for a parity of 0 the target is -0.5. The objective function is the sum of squares error for the 128 training patterns. Since this is a minimisation problem, it is transformed into a maximisation problem for use by the ACO algorithm by negating it.

The network contains a bias unit whose input is permanently set to 1.0. The bias unit is connected to the 5 hidden units and single output unit. Each input unit is connected to the 5 hidden units, and each hidden unit is connected to the output unit. The network topology is shown in Figure 5 (only the first input unit's connections are shown).

The values of the weights are restricted to the range [-10, 10]. 8 bits are used to represent each weight in Baluja (1995), and the DACO is applied using the same discrete representation (ie. [0, 255]). All hidden and output units use a sigmoid activation function *(tanh(activation)/2),* with no scaling of the input activation.

The results obtained, with 200,000 evaluations for each method, are shown

Method	Average Maximum
DACO	-7.84
PBIL (Gray code)	-8.2
SGA (Gray code)	-11.6
Hill-Climbing (Gray code)	-13.7
CACO	-8.89

Table 9. Results for ANN parity network

in Table 9. The results for the other methods have been negated since the objective function was negated for the DACO.

4.2. Constrained problems

The ant colony search model for constrained optimisation is first tested on the five test cases proposed by Michalewicz (1995b). During these experiments the constraint violation is calculated as Eucleadian distance from the accepted feasible region.

test case #1: $F(X) = 5.1x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i - \sum_{i=5}^{13} x_i$

subject to:

$2x_1 + 2x_2 + x_{10} + x_{11} \le 10,$
$2x_1 + 2x_3 + x_{10} + x_{12} \le 10,$
$2x_2 + 2x_3 + x_{11} + x_{12} \le 10,$
$-8x_1 + x_{10} \le 0,$
$-8x_2 + x_{11} \le 0,$
$-8x_3 + x_{12} \le 0,$
$-2x_6 - x_7 + x_{11} \le 0,$
$-2x_4 - x_5 + x_{10} \le 0,$
$0 \le x_i \le 1, i = 1, \ldots, 9, 13,$
$0 \le x_i \le 100, i = 10, 11, 12.$

The problem has 9 linear constraints; the cost function is quadratic with global minimum at $X = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$, where $F(X) = -15$.

test case #2: $F(X) = x_1 + x_2 + x_3$

subject to:

$1 - 0.0025(x_4 + x_5) \ge 0,$
$1 - 0.0025(x_5 + x_7 - x_4) \ge 0,$
$1 - 0.01(x_8 - x_5) \ge 0,$
$x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333 \ge 0,$
$x_2x_7 - 1250000 - x_3x_5 + 2500.1x_5 \ge 0,$
$100 \le x_1 \le 10000,$
$1000 \le x_i \le 10000, i = 2, 3,$

$10 \leq x_i \leq 1000$, i = 4, ... , 8.

The problem has 3 linear and 3 non-linear constraints; the cost function is linear and has its global minimum at X = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979) where $F(X)$ = 7049.330923.

test case #3: $F(X) = (x1 - 10)^2 + 5(x_2 - 12)^2 + x_i + 3(x_4 - 11)^2 + lOx + 7x + x* - 4x6x7 - 10x6 - 8x1$

subject to:

$127 - 2Xi - 3X - X3 - 4X - 5X5 \geq 0$,

$282 - 7x1 - 3x_2 - 10x5 - X4 + X5 \geq 0$,

$192 - 23x1 - x\$ - 6x + 8x1 \geq 0$,

$-4xr - X\$ + 3x1x_2 - 2x5 - 5r6 + 11x7 \geq 0$,

$-10.0 \leq x_i \leq 10.0$, i = 1, ... , 7.

The problem has 4 non-linear constraints; the cost function is non-linear and has its global minimum at X = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227) where $F(X)$ = 680.6300573.

test case #4: $F(X) = e^{x_1 x_2 x_3 x_4 x_5}$

subject to:

$Xi + X\$ + X5 + X + X = 10$,

$X_2X3 - 5X4X5 = 0$,

$xr + x = -1$,

$-2.3 \leq x_i \leq 2.3$, i = 1,2,

$-3.2 \leq x_i \leq 3.2$, i = 3,4,5.

The problem has 3 non-linear constraints; the cost function has its global minimum at X = (-1.717143, 1.595709, 1.827247, -0.7636413, -0.7636450) where $F(X)$ = 0.0539498478.

test case #5:

$$F(X) = Xi + x + XIX_2 - 14x1 - 16x_2 + (x3 - 10)^2 + 4(x_4 - 5)^2 + (x5 - 3)^2 + 2(x6 - 1)^2 + 5x + 7(xs - 11)^2 + 2(x9 - 10)^2 + (x10 - 7)^2 + 45$$

subject to:

$105 - 4x1 - 5x_2 + 3x1 - 9xs \geq 0$,

$-10x1 + 8x_2 + 17x1 - 2xs \geq 0$,

$8x1 - 2x_2 - 5xg + 2x10 + 12 \geq 0$,

$-3(x1 - 2)^2 - 4(x_2 - 3)^2 - 2x5 + 7x4 + 120 \geq 0$,

$-5xr - 8x_2 - (x3 - 6)^2 + 2x4 + 40 \geq 0$,

$-Xi - 2(x_2 - 2)^2 + 2X1X_2 - 14X5 + 6X6 \geq 0$,

$-0.5(x1 - 8)^2 - 2(r,_2 - 4)^2 - 3r, + X6 + 30 \geq 0$,

$3x1 - 6x_2 - 12(xg - 8)^2 + 7x10 \geq 0$,

$-10.0 \leq x_i \leq 10.0$, i = 1, ... , 10.

The problem has 3 linear and 5 non-linear constraints; the cost function has its minimum at X = (2.171996, 2.363883, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927) where $F(X)$ = 24.3062091.

runs	test 1	test 2	test 3	test 4	test 5
opt.	-15.00	7049	680.6	0.054	24.30
1	-14.39	7293	680.8	0.053	26.53
2.	-14.46	7624	680.8	0.054	25.83
3.	-14.76	7486	680.9	0.055	25.81
4.	-14.45	7674	680.8	0.066	26.04
5.	-13.93	8009	680.8	0.056	26.12
6.	-14.59	7343	681	0.055	25.78
7.	-14.46	7774	681	0.057	26.78
8.	-14.53	7650	680.8	0.055	25.72
9.	-14.33	8082	680.9	0.054	26.29
10.	-14.64	7704	681	0.056	25.72
aver.	-14.45	7663	680.9	0.056	26.06

Table 10. Results of running the Ant Colony model on the five test cases proposed in Michalewicz (1995b). The assumed constraint violation accuracy is 0.01 for each constraint.

The ant colony search model outperforms six of the existing state-of-the-art evolutionary constrained handling techniques described in Michalewicz (1995b). Results arc summarised in Table 10.

Compared to other evolutionary methods for constrained optimisation the ant colony model shows excellent performance and quality of solution especially for the problems with non-linear constraints. A remarkable feature of the ant colony model is that the standard deviation of the solutions (averaged over 10 independent runs) is considerably less that the standard deviation produced by other evolutionary methods for constrained optimisation Michalewicz (1995b).

4.2.1. Keane's Bump (20D, 50D)

Keane (1994) bump problem is defined for any number of dimensions N, although $N = 20$ and $N = 50$ arc commonly used instances. This problem is of particular interest to engineers since it resembles many engineering optimisation problems: it is highly multimodal, has high dimensionality ($N = 50$), and the global maximum is found against the constraint boundary.

The function is:

$$f(\bar{x}) = \frac{\left|\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)\right|}{\sqrt{\sum_{i=1}^{n} i x_i^2}}$$

Method	Evaluations	Average Maximum
DACO	30,000	0.826
GA	150,000	0.779
PBIL	150,000	0.755
EP	150,000	0.673
ES	150,000	0.578
SA	150,000	0.395
CACO	30,000	0.823

Table 11. Results for Keane's Bump 50D

Subject to the constraints:

$$\prod_{i=1}^{n} X_i > 0.75, \quad \sum_{i=1}^{n} X_i < \frac{15n}{2}$$

where n is the number of dimensions.

While the global maximum is not immediately obvious, the best achieved value using ACO is 0.834916.

In order to facilitate comparison with results obtained using GA and PEIL with 12-bit precision, the DACO algorithm is used for comparison with a discrete range of [0, 4095].

For the Bump problem in 50 dimensions, the results obtained are shown in Table 11.

4.2.2. Summary of results for test functions

This section summarises and discusses the results obtained for the test functions given in the previous section.

Table 12 shows the ACO ranking for each test function, and which version of the ACO algorithm (DACO or CACO) performed best. For test functions where the other methods used for comparison used discrete coding, only the DACO result is used for ranking since the CACO result cannot fairly be compared.

4.3. Discussion of results

As is seen from Table 12, the ACO algorithm performs very well for all functions, except Baluja's Function 3. While for this function, the ACO algorithm does not perform as well as PEIL and Multiple-Restart Hill-climbing, it still outperforms the Genetic Algorithm. Although not taken into account for ranking of the algorithms, the Continuous ACO performs extremely well for this function. (See the Appendix).

Test Function	ACO Ranking	Best ACO Version
De Jong F2	1	DACO
De Jong F3	1	DACO
Keane's Bump 50D	1	DACO
Griewangk	1	DACO
Homaifar Function 1	1	CACO
Homaifar Function 2	1	CACO
Baluja F1	1	DACO
Baluja F2	1	DACO
Baluja F3	3	CACO
ANN Weight Optimisation	1	DACO

Table 12. Summary of test results

The reason for this improved performance is most likely to be due to the increased resolution of the CACO, and not due to more effective search of the problem space. Baluja has indicated that with increased accuracy, the Hill-Climbing algorithm obtains higher values than those found using CACO [personal correspondence]. The use of Gray coding leads to improved performance of the other algorithms, whereas the DACO uses a discrete representation that does not rely on the bit coding, and hence cannot benefit.

For the remaining two of Baluja's test problems, the discrete ACO outperforms the other algorithms, and performs better than the continuous ACO. While initial portions of the search for DACO and ACO arc reasonably similar, the DACO exhibits the ability to find improved fitness values up to the end of the search process. The CACO's progress, however, stagnates and is unable to improve the fitness value for the last half of the search. Since the Crossover Probability used was 0.0, the final stages of the search consist almost entirely of local search. While the discrete local search process improves the fitness, the continuous local search process is unable to. This indicates that the continuous local search algorithm is of little benefit for these functions, and may be of little use in the continuous search process.

The crossover process is detrimental to the performance of the ACO for Baluja's Function 1 and Function 2, as explained in the. previous paragraph. The selection of a suitable Crossover Probability is made difficult since in most cases the objective function is not known mathematically, but is the result of a simulation or experiment. As a compromise, if the level of coupling in an objective function is unknown and cannot be determined, a value of 0.5 is a good compromise. Using CP=0.5 for Baluja's Function 1 and 2, gives an average fitness of 2.4536 and 8.7965 respectively (with xlO0 scaling). Thus, performance is on a par or better than existing algorithms, even using the non-optimal setting

from CP.

In summary, the results indicate that the ACO algorithm is a powerful optimisation tool. A useful optimisation algorithm should not only be powerful, but simple to use. It should not require fine-tuning of parameter values, since this is usually a time-consuming task.

5. Conclusions

The results obtained for the empirical comparison indicate that the ACO algorithm is as good, if not better, than methods such as GA, PEIL, Hill-climbing and Evolutionary Programming for the problems tested. As the range of problems is broad, with different types of problems being well represented, the ACO's performance on these problems suggests it will perform well in general.

The excellent performance of the algorithm for most problems using the default parameter values suggests that little parameter-tweaking is necessary in general. For problems that exhibit a large degree of inter-dependence between variables in the solution vector, a low Crossover Probability is optimal, although the algorithm using the default value outperforms standard Genetic Algorithms on such problems.

The ability to apply the algorithm using real or discrete coding is advantageous, since some problems are more naturally represented and more easily solved using real coding. However, since many problems in engineering design contain discrete variables, a version of the ACO allowing mixed real and discrete variables may be highly useful.

In summary, the ant colony metaphor is a powerful optimisation paradigm, providing an effective framework for the communication between two methods in a hybrid optimisation algorithm. The ACO algorithm is similar to hybrid genetic algorithms with local search, and the extensive use of GA-like operations makes the naming of the algorithm debatable. It could more correctly be called an Ant-Genetic-Algorithm to illustrate the various methods employed during the optimisation process.

The ant colony metaphor is a powerful paradigm for creating hybrid optimisation techniques, since the application of local search agents is well covered. In traditional hybrid methods (like Genetic Algorithms with Local Search, GA-LS), the criterion for applying local search is less well-defined. The ACO algorithm is also powerful in that it allows local agents to co-operate with other, while at the same time communicating information to the global agents. Other methods typically lack such co-operation between local agents. The food-source exhausting analogy is useful in focusing local search in areas where improvement is most likely, which efficiently distributes the local search agents through the search space.

The algorithm has some disadvantages, specifically the large number of control parameters. Some parameter values arc problem-specific, making the algorithm less attractive for use when compared to simpler methods such as PEIL.

However, in most cases, the default parameter values allow the algorithm to perform well in general.

Further avenues of research include studying other hybrid combinations within the ACO framework, for example, using PBIL instead of GA-like methods for global search, utilising a more powerful local search algorithm, such as Sequential Quadratic Programming, and the application of the ACO algorithm to real-world engineering design problems.

References

BALDJA, S. (1994) *Pop'll,lation-Based Incremental Learning: A Method for Integrating Genetic Search Based Fv,nction Optimization and Competitive Learning.* Internal paper, CMTT-CS-94-163, School of Computer Science, Carnegie Mellon University.

BALDJA, S. (1995) *An Empirical Comparison of Seven Iterative and Evol'll,tionary F'll,nction Optimization Hev,ristcis.* Internal paper, CMU-CS-95-193, School of Computer Science, Carnegie Mellon University.

BILCHEV, G. and PARMEE, I. (1995) The ant colony metaphor for searching continuous design spaces. In: Fogarty, T., ed., *Lecfore Notes in Compv,ter Science,* 993, Springer Verlag, 25-39.

BILCHEV, G. and PARMEE, I. (1996) Constrained optimization with and ant colony search model. *Prncs. of the ACEDC,* Plymouth, UK, 145-151.

CoLORNI, A., DoRIGO, M. and MANIEZZO, V. (1991) Distributed optimization by ant colonies. *Prncs. of the Em'Opean Conference on Artificial Life (ECAL),* Paris.

DE JONG, K. (1975) An Analysis of the Behavior of a Class of Genetic Adaptive Systems, PhD thesis, University of Michigan, Ann Arbor.

DoRIGO, M., MANIEZZO, V. and CoLORNI, A. (1996) The ailt system: optimisation by a colony of cooperating agents. *IEEE Transaction on System, Man and Cybernetics,* B, 26, 1, 1-13

FOGEL, D. (1995) A comparison of evolutionary programming and genetic algorithms on selected constrained optimization problems. *Sim'll,lat?:on,* 64, 4, 397-404.

HART, W. (1994) *Adaptive Global Optimization with Local Search.* PhD thesis, University of California, San Diego.

KEANE, A. (1994) Experiences with Optimisers in Structural Design. *Procs. of the ACEDC,* Plymouth, 14-27.

MICHALEWICZ, Z. (1995A) *Genetic Algorithms + Data Strncfores = Evolv,tionary Programs.* 3rd Edition, Springer-Verlag.

MICHALEWICZ, Z. (1995B) *A Śv,rvey of Contraint Handling Techniqv,es in Evol11,tionary Computation Methods.* 4th Anual Conference on Evolutionary Programming, San Diego.

WODRICH, M. (1996) *Ant colony optimisation : An empirical investigation into an ant colony metaphor for continuous function optimisation.* Un-

dergraduate Thesis, University of Cape Town.

Appendix: Plots of ACO Search Progress

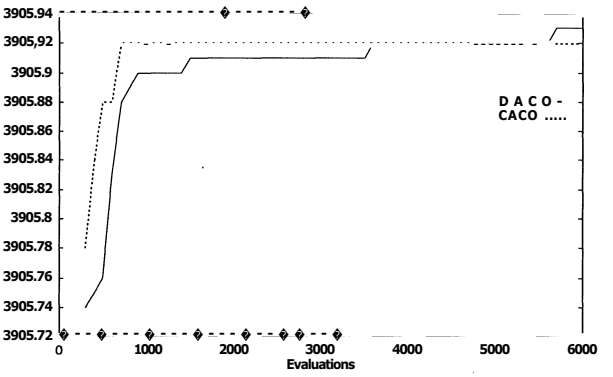

Figure 8. De Jong F2 (Average Fitness over 10 runs)

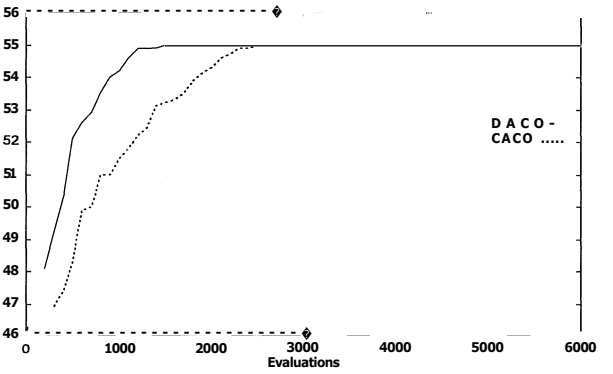

Figure 9. De Jong F3 (Average Fitness over 10 runs)

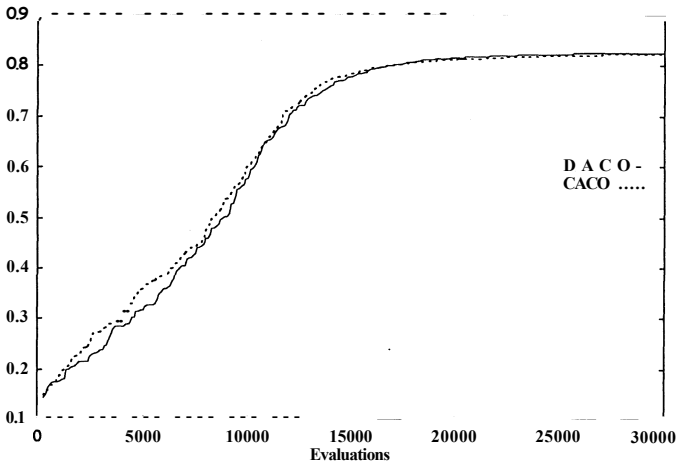

Figure 10. Keane's Bump 50D (Average Fitness over 10 runs)

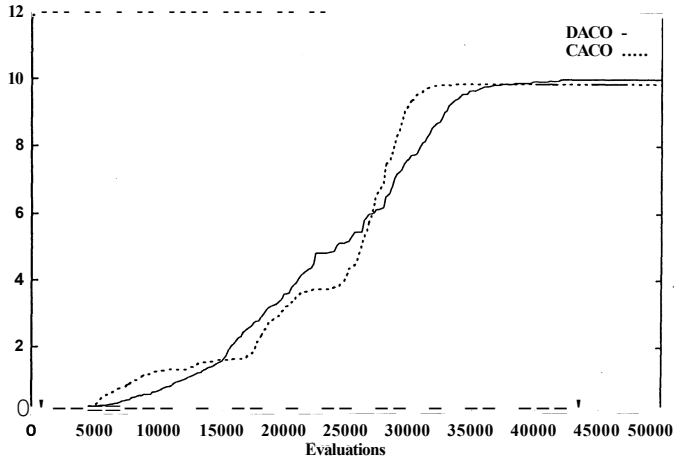

Figure 11. Gricwangk (Average Fitness over 10 runs)

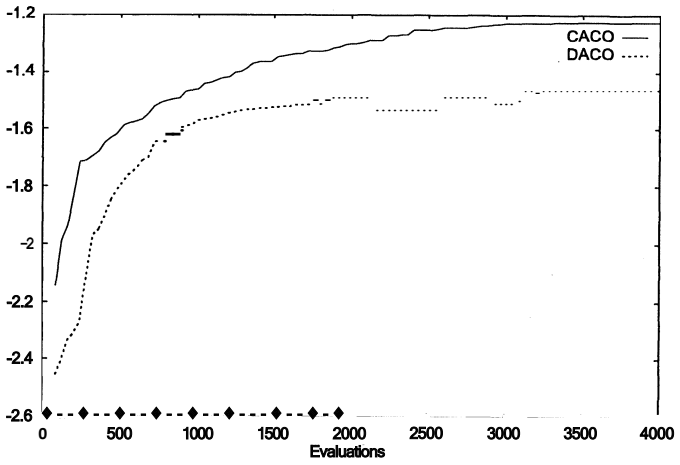

Figure 12. Homaifar Function 1 (Average Fitness over 10 runs using CACO)

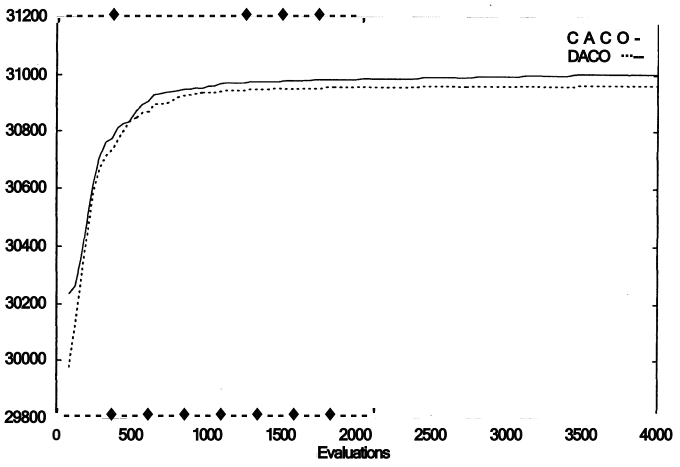

Figure 13. Homaifar Function 2 (Average Fitness over 10 runs using CACO)

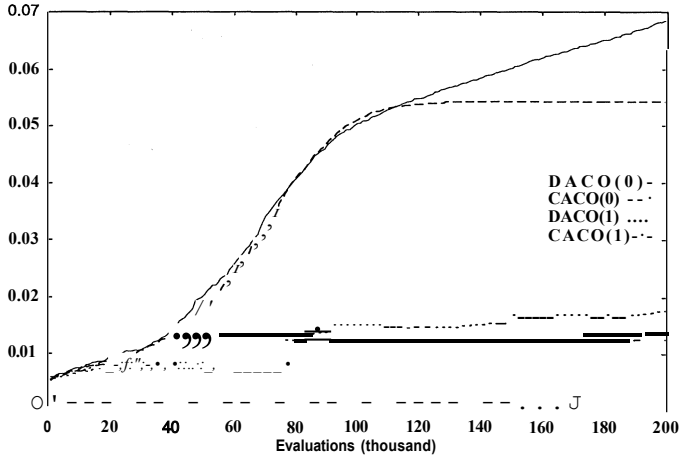

Figure 14. Baluja F1, comparing *GP* of Oand 1 (Average Fitness over 10 runs)

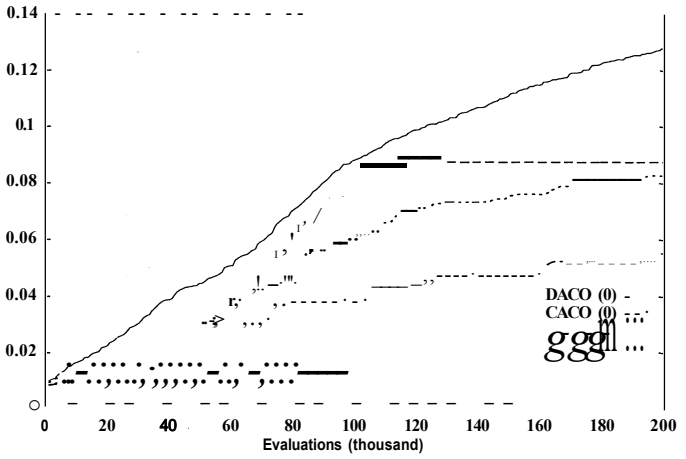

Figure 15. Baluja F2, comparing *GP* of Oand 1 (Average Fitness over 10 runs)

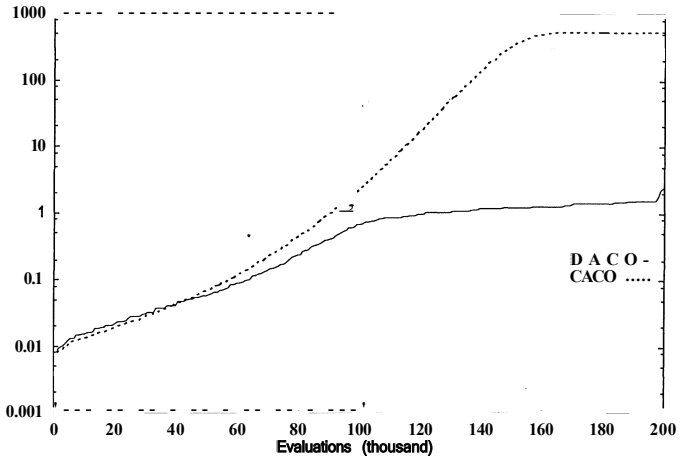

Figure 16. Baluja F3 (Average Fitness over 10 runs)

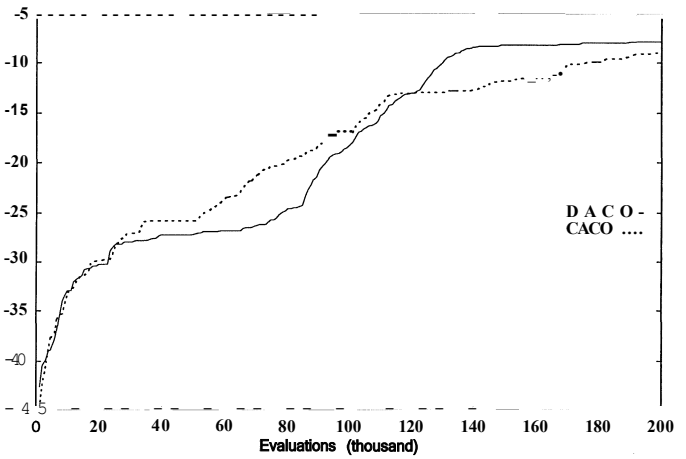

Figure 17. ANN Weight Optimisation (Average Fitness over 10 runs)