MODIFIED CONTINUOUS ANT COLONY ALGORITHM

A. Aidov, G. S. Dulikravich

Department of Mechanical and Materials Engineering,
MAIDROC Lab., Florida International University, FIU, 10555 West Flagler St., EC 3474,
Miami, Florida 33174, U.S.A.
e-mail: aaido001@fiu.edu; dulikrav@fiu.edu

Abstract. This relatively new concept in optimization involves the use of artificial ants and is based on real ant behavior inspired by the way ants search for food. In this paper, a novel ant colony optimization technique for continuous domains is presented. The goal was to provide improvements in computing time and robustness when compared to other optimization algorithms. The developed Modified Continuous Ant Colony (MCACO) algorithm was run for numerous classic test cases for both single and multi-objective problems. The results demonstrate that the method is robust, stable, and that the number of objective function evaluations is comparable to other optimization algorithms.

1. Introduction

There are two types of optimization problems. The first type has a single objective and the second type of problems has multiple objectives. Single-objective optimization has the goal of finding the global minimum of the possible multi-extremal function of one or more independent variables. The goal of multi-objective optimization is to find a Pareto set of non-dominated solutions representing the best possible trade-offs of multiple simultaneous objectives. Optimization methods consist of a broad spectrum of optimization algorithms that can be grouped into the following categories: gradient based and non-gradient based algorithms. The method exposed in this thesis involves the technique of Ant Colony Optimization (ACO) and belongs to the category of non-gradient based methods.

The ACO scheme was first proposed by Marco Dorigo in 1992 [1]. He noted that ants communicate through a form of stigmergy in that they lay trails of chemical substances called pheromones as they scurry around in search of food [2]. These chemical trails can be followed by other ants. At its roots, the ACO algorithm is a metahueristic for combinatorial optimization [2]. However, when dealing with continuous spaces, such as those found in function optimization, the ACO metahueristic does not work. The ACO routine is mainly used for discrete space problems, for example the traveling salesman problem and the quadratic assignment problem [3].

Numerous researchers proposed extensions of the ACO metahueristic to continuous space problems. One such extension is called Continuous Ant Colony Optimization (CACO). It was first envisioned by Bilchev and Parmee in 1995 [4]. It involves the use of a simulated nest from which an ant moves in a certain calculated direction to a specified distance [5]. The goal of this research is to create and modify a CACO algorithm so that it requires less computing time and has better robustness compared to other optimization

algorithms. Hence, the name of the novel technique is Modified Continuous Ant Colony Optimization (MCACO). MCACO is also extended to multi-objective optimization problems with the help of the Normalized Normal Constraint (NNC) method. This method should help obtain a set of optimal solutions that are equally distributed along the Pareto frontier. The main concepts that make up the MCACO code include random number generation for selection of search direction, ant movement distance, fitness evaluation of the objective function, pheromone update, and search radius update. The code is written using the C++ computer language and utilizes the Mersenne Twister random number generator developed by Matsumoto and Nishimura [6]. Pheromone density plays a vital role in the MCACO algorithm as it directly affects the direction an ant chooses to proceed in. Some important modifications that are researched include ant movement alterations and search radius reduction techniques. Classical test functions for both single-objective and multi-objective cases are used to evaluate the performance of the algorithm.

2. Theoretical Concepts

2.1. Single-objective optimization

The goal of the single-objective optimization problem is to find the single global minimum over a desired search space. The single-objective optimization problem can be formulated as [7],

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad x \in \Omega \end{aligned} \tag{1}$$

The function f(x) is called the objective function while the vector x is a vector of n independent variables denoted by $x = \left[x_1, x_2, ..., x_n\right]^T$. The variables themselves, the $x_1, x_2, ..., x_n$ values, are called the design variables. When $\Omega = R^n$, this problem is denoted as the general form of single-objective unconstrained optimization [7]. However, when Ω is only a proper subset of n-dimensional Euclidean space, written as $\Omega \subset R^n$, the problem may be formulated as [8],

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad x \in \Omega \\ &c_i(x) = 0, \quad i \in E \\ &c_i(x) \geq 0, \quad i \in I \end{aligned} \tag{2}$$

The set Ω is now called the constrained set or feasible region. Equation (4) is a prime example of a single-objective constrained optimization problem. Note that the $c_i(x)'s$ are constraint functions, while E and I represent the index sets of equality and inequality constraints [8]. The reason why equations (1) and (2) are formulated as minimization problems is because minimizing a function $f(x)$ is equivalent to maximizing $-f(x)$ [7]. As a general rule, optimization problems are usually defined as minimization problems.

2.2. Combinatorial optimization

A combinatorial, or discrete, optimization problem is a problem that has a feasible search region which is discrete. The variables used for the objective functions are assumed

to be of a discrete type, such as integers. The basic model of a combinatorial optimization problem is given below [9].

$$\begin{aligned} &\text{A model } P = (S, \Omega, f) \\ &\text{A search space } S \text{ defined over a finite set of discrete variables} \\ &\text{A set } \Omega \text{ of constraints among variables} \\ &\text{An objective function } f \text{ to be minimized} \end{aligned} \tag{3}$$

The most famous combinatorial optimization is the Traveling Salesman Problem (TSP). TSP is the problem of a salesman, who has to find the shortest possible trip through a group of cities, while visiting each city only once and returning home. The TSP can be represented as a complete weighted graph. Essentially, solving the TSP requires finding the minimum length Hamiltonian cycle of the graph.

2.3. Ant Colony Optimization (ACO)

One of the most important topics in ACO algorithm is the concept of stigmergy which is defined as an indirect communication via interaction with the environment [2]. An example of this idea can be shown between two ants. Two ants can interact indirectly when one of the ants alters the environment and the other ant reacts to the new environment later on [10]. The stigmergy concept can be described by the idea of pheromones.

Many real ant species, such as the *Linepithema humile*, deposit on the ground a substance called pheromone, as they travel to and from a food source. Other ants searching for food can sense the pheromone and have their movements influenced by its strength. The concept of pheromones can by explanted with the figure 1.

Part (a) shows the nest, the food source, and the ants traveling between the nest and the food. In part (b), an obstacle is placed between the nest and the food source. As a result, in part (c), the ants travel around the obstacle in both directions. Note that the eastern path around the obstacle is much shorter than the western path. Because there are more ants on the eastern path around the obstacle, the pheromone concentration in that direction accumulates faster than the pheromone density in the western direction [9]. Over time, the rest of the ants follow the eastern path between the nest and the food source as shown in part (d). Pheromone trail intensity is proportional to the utility of using that specific trail to build quality solutions. Pheromone evaporation is another important concept in ACO. It simulates the realistic decreases of pheromone intensity over time if a particular trail is not used [2].

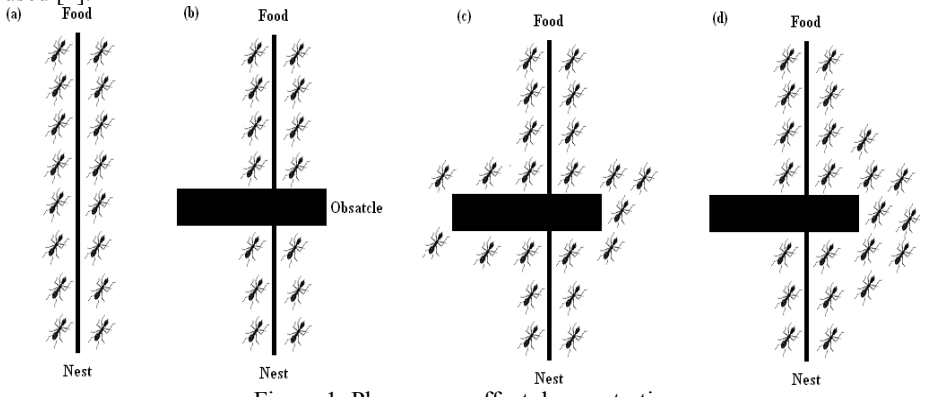

Figure 1: Pheromone effect demonstration

The ACO metahueristic pseudo-code is given in the algorithm that follows [9]:

```
Set parameters, initialize pheromone trails
While termination conditions not met do
    ConstructAntSolutions
    ApplyLocalSearch
    UpdatePheromones
End while
```

(4)

The three important procedures in the ACO metahueristic are ConstructAntSolutions, ApplyLocalSearch, and UpdatePheromones. The first procedure constructs solutions from elements of a finite set of solution components using a number of artificial ants. The ants move by applying a stochastic local decision procedure that is weighted by pheromone trails and heuristic information [2]. The next procedure implements problem-specific measures and performs centralized actions. The final procedure in the ACO metahueristic increases pheromone values associated with good solutions and decreases those that are associated with bad ones. The addition of pheromone concentration makes it likely that future ants will use the same connections [2]. In the ACO algorithm, artificial ants construct solutions by moving through construction graphs or discrete connected data points [9]. Using the ACO algorithm in combination with the ideas of stigmergy, many interesting problems can be solved [3]. To sum up, ACO can be viewed as a metahueristic in which artificial ants work together to find optimized solutions to discrete optimization problems [2]. As a result, the ACO algorithm cannot be used to optimize continuous optimization problems because the algorithm is only prescribed for discrete optimization problems.

2.4. ACO algorithms for continuous domains

Researchers have extended ACO ideas to problems with continuous domains, such as in the optimization of functions. Table 1 lists some of the ant colony based methods that are applicable to continuous problems:

Table 1: Continuous ant based optimization techniques

Method name	Reference	Abbreviation
Continuous Ant Colony Optimization	[11]	CACO
Continuous Interacting Ant Colony	[12]	CIAC
Direct Ant Colony Optimization	[13]	DACO
ACO extended to continuous domains	[14]	ACO_R

CACO is based on a local search in the vicinity of a nest [11]. CIAC is based on the construction of a network of ants that are set up through a heterarchical manner and it uses dual communication channels for ants to exchange information [12]. DACO is an algorithm based on using a pheromone definition and an update rule which is directly associated with the mean and deviation value of a specific normal distribution [13]. ACO_R is co-developed by the original architect of the first ACO algorithm, Marco Dorigo. This method

uses a probability density function to sample points [14]. The ant based algorithm for continuous space constructed and modified in this thesis is CACO.

2.5. Continuous Ant Colony Optimization (CACO)

CACO was the first ant colony based technique developed that was suitable for continuous function optimization [15]. The main difficulty in applying any ant colony optimization algorithm to continuous problems is to model a continuous domain with a discrete data structure. In the original ACO routine, the ants wade through a network of connected nodes to find a solution. However, in the continuous case, there is no network of nodes, but just a continuous space instead. This difficulty is solved by using a starting base point called the nest. The nest is the structure where ants begin the search from. A finite number of search directions, represented as vectors, emanate from the nest [4]. The ants, during every iteration of the CACO algorithm, choose one of the vectors to follow probabilistically [14]. The pseudo-code for the CACO algorithm is outlined below [4]:

```
begin
   t ← 0
   initialize   A(t)
   evaluate   A(t)
   while  (not  end_cond)  do
   begin
      t ← t + 1
      add_trail   A(t)
      send_ants   A(t)
      evaluate   A(t)
      evaporate   A(t)
   end
end
```
(5)

The function A(t) is the data structure representing the nest and its vicinity [4]. The first step is to initialize the nest structure by generating random starting search direction vectors. Next, the search radius is defined. This value determines the maximum distance that an ant can move at a single time. Then, “initialize A(t)” sends ants in various search directions while “evaluate A(t)” calls the objective function evaluation. The command “add_trail” is synonymous to the ants laying pheromones on the trails. This is the basic version of the CACO algorithm [11].

When a chosen search direction does not result in improvement of the objective function, it is not taken into consideration in the trail adding process. Actually, the reverse occurs in this case and the pheromones evaporate. This is analogous to food exhaustion in a real ant colony. Many of the ideas, including those of stigmergy and pheromones, are borrowed directly from the ACO algorithm to be used in the CACO algorithm.

When CACO was first developed, it was intended for the local search portion of a global optimization. In effect, CACO was used in conjunction with a genetic algorithm or other type of global optimizer to reach a satisfactory point for local exploration. However, this approach was later expanded to include global search as well. One of the ways to apply this technique as a global optimization algorithm is to first divide the domain into a specific

number of regions. These regions would then serve as the local stations from which the ants would venture out and explore [15].

Although CACO draws inspiration from the original ACO algorithm, it does not follow it exactly. One of the major differences is the idea of the CACO nest, as there is no nest in the ACO algorithm. Another key difference is the idea of an incremental construction of solutions. In ACO, solutions were constructed incrementally to be able to solve combinatorial optimization problems such as the TSP. However, CACO is used for continuous problems and makes no use of a buildup of solutions. Although the methods described have been applied to single-objective optimization problems with success, the solution of multi-objective problems is a different matter.

2.6. Multi-objective optimization

Unlike in the case of a single-objective problem, a multi-objective problem has several objectives which need to be optimized simultaneously. In single-objective optimization, there is only a single search space called the decision variable space. However, in multi-objective problems there is, in addition to decision variable space, an entity called objective space. The relation between these two spaces is defined by the mapping between them. Even so, the mapping is often complicated and nonlinear. Not only are the properties of the two spaces often dissimilar, but also a small perturbation in one space can result in an immense change in the other [16].

The general multi-objective optimization problem can be stated as follows [17],

$$\begin{aligned} &\underset{x}{\text{Minimize}} \quad F(x) = \left[F_1(x), F_2(x), ..., F_k(x)\right]^T \\ &\text{subject to} \quad g_j(x) \le 0, \quad j = 1, 2, ..., m \\ &\qquad\qquad h_l(x) = 0, \quad l = 1, 2, ..., e \\ &\qquad\qquad x_l \le x \le x_u \end{aligned} \tag{6}$$

The value k represents the number of objective functions. Since k must always be ≥ 2, the name of the problem is multi-objective optimization. The variables m and e symbolize the number of inequality constraints and equality constraints, respectively [17].

The most important concept in multi-objective optimization is called Pareto optimality. As a result of there being many objectives that are often conflicting, there is no single correct solution. In multi-objective optimization problems, solutions are sought where none of the objectives can be improved without the worsening of at least one of the other objectives. These are called Pareto optimal solutions and they form a hyper-surface in the objective function space. In more general terms, the definition is given below [18],

$$\begin{aligned} &\text{Assume } S \text{ is feasible region,} \\ &\text{A decision vector } x^* \in S \text{ is Pareto} \\ &\text{optimal if there does not } \exists \\ &\text{another decision vector } x \in S \text{ such that} \\ &f_i(x) \le f_i(x^*) \text{ for } \forall \text{ i and} \\ &f_j(x) < f_j(x^*) \text{ for at least one j} \end{aligned} \tag{7}$$

There are theoretically an infinite number of Pareto optimal solutions for every multi-objective optimization problem [18]. These points form a Pareto frontier which is a hyper-surface whose dimensionality is the same as the number of the simultaneous objectives. The Pareto frontier does not have to be a smooth surface and can have discontinuities.

2.7. Methods for multi-objective optimization

There are many methods that are used to solve multi-objective optimization problems. A few of the methods include tabu search and weighting method [19]. Tabu search is a metahueristic that is based on the idea that to rate the quality of a solution to a problem as intelligent, it must make use of adaptive memory and sensible, responsive exploration [19]. One of the most simple and often used multi-objective optimization techniques is called the weighted global criterion method. In this method, all of the objectives are combined to form a single-objective function which can be optimized using single-objective optimization techniques. Table 2 below shows three of the most popular weighted methods [17]:

Table 2: Weighted global criterion methods

Weighted Sum Method	$U = \sum_{i=1}^{k} w_i F_i(x)$
Exponential Weighted Criterion	$U = \sum_{i=1}^{k} \left(e^{p\ w_i} - 1\right)\ e^{p\ F_i(x}$
Weighted Product Method	$U = \prod_{i=1}^{k} \left[F_i(x)\right]^{w_i}$

In the table above, U represents final combined single-objective function and w_i represents the weights used. The most popular weighted criterion method, by far, is the weighted sum method. The weighted sum method is also an excellent method to use in combination with a continuous-type ant colony optimization algorithm to obtain the Pareto frontier. However, the weighted method has a few major deficiencies. This method only works for continuous convex Pareto frontiers.

If the Pareto frontier is concave, there are no possible combinations of weights for which the solution would be graphed to the concave part. Another failure of the weighted sum method is the fact that it does not work if the Pareto curve has discontinuities. An additional deficiency is that an even spread of points on the Pareto frontier cannot be created by an even spread of weights [20].

A solution to the curvature deficiency would be to use the weighted compromise method [21] given as

$$\text{minimize } f(x)=\sum_{i=1}^{m} w_i \left(f_i(x)\right)^{c_i} \tag{8}$$

Altering the c_i exponent value manipulates the function topology and increases the curvature. As a result, the method is able to capture points on the concave part of the Pareto curve. However, the exact exponent value that is needed to capture all of the Pareto points is generally unknown. The weighted compromise method also suffers from the difficulty of producing an even spread of points for an even set of weights. Special methods and clustering techniques are therefore used to make sure the allocation of Pareto optimal

solutions are equally distributed. One such method, invented by Messac [22] is called the Normalized Normal Constraint (NNC)method.

2.8. Normalized Normal Constraint (NNC) Method

The NNC method can generate an evenly distributed set of Pareto solutions and is valid for both convex and concave functions. Basically, this technique fixes the problems associated with the weighted sum method. The NNC method works by performing a series of optimizations where each optimization is subject to a reduced feasible design space [22]. With every design space reduction, one Pareto optimal point is obtained. This is done by transforming the original multi-objective problem into a single-objective problem and by minimizing the single-objective problem which is subject to the reduced feasible space. The NNC method starts out with the original entire design space and it reduces the entire design space until the space has been completely explored. This approach allows the method to generate Pareto solutions throughout the whole Pareto frontier [22].

Under certain uncommon circumstances, the NNC method can generate non-Pareto and weak Pareto solutions [23]. When the aforementioned occurs, a Pareto filter can be used [23]. It is an algorithm that eliminates all dominated points from the solution set [24]. To avoid another pitfall related to scaling deficiencies, the optimization is performed in the normalized objective space [25]. Having addressed the issues at hand, the methods behind the MCACO algorithm can now be explained.

3. Modified Continuous Ant Colony Optimization (MCACO)

The MCACO algorithm is built using the same principles as the CACO algorithm. The CACO nest is used as well as pheromone values to guide the ants. New features that have been developed include the multiple nest technique and the mobilization of the nest location. The search direction pattern used is also a new feature that was introduced in MCACO. The single-objective version of the MCACO algorithm can be broken up into two main parts as shown in figure 2.

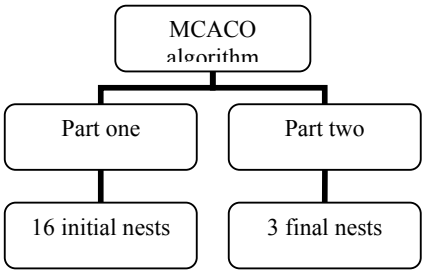

Figure 2: MCACO algorithm

Part one of the algorithm features sixteen initial nests spread across the search domain. Part two of the MCACO routine features three final nests. To sum up the algorithm briefly, ants begin at the nests and move around the search space in certain directions looking for minimum fitness values of the functions to be optimized. The goal of part one of the MCACO algorithm is to locate general areas of minimum fitness and get close to the global

minimum. The goal of part two is to thoroughly explore the areas of minimum fitness and find the global minimum.

Once part one of the algorithm completes running, the nests are ranked in order of best minimum values obtained. The three nests with the lowest minimum fitness values are selected. At the location of each of the three best minimums, a new nest is initialized and part two starts to run. Part two of the algorithm searches around the final three partially optimized nest locations. The location and value of the minimum of the three final nests is considered the global minimum solution.

3.1. Ant and nest movement

Each ant located at each nest has the capacity to move in four search directions. In part one of the algorithm, the four search directions alternate between two different sets of search directions. Set one uses the four directions situated at 0, 90, 180, and 270 degrees. Set two uses the four search directions located at 45, 135, 225, and 315 degrees. Every time a certain number of function evaluations are completed, the set alternates between set one and set two. This scheme lets the ants explore the search space in a structured manner through a possible eight different search directions.

Another important topic related to ant movement is the shrinkage of the search radius over time. As the algorithm runs its course, the movement of the ants is restricted more and more. Over the course of the algorithm, the ants are able to narrow down on the global minimum.

An additional key topic is the idea of the nests moving to new locations. The nests are also not stationary and shift around the search space to new locations over time. At the beginning of the MCACO algorithm, the fitness value at each nest is evaluated. This fitness value is held as the best current known minimum for a specific nest. As the ants explore out from each nest, they find new minima with lower values of the function fitness. These new minima are the locations to which the nests move to.

In part two of the algorithm, the search direction that an ant can choose when the three final nests are selected is chosen at random. This is a different direction selection process than that used in part one of the algorithm, where a set structure was used. The search directions are randomized so that the ants can have more freedom to search for the global minimum in the second part of the algorithm.

3.2. Search direction selection

In the MCACO algorithm, the search direction is selected through the roulette wheel concept (figures 3 and 4). Assume each of the numbered pieces of the wheel to represent one of the possible four search directions. Initially, as shown in figure 3, the size of each piece is the same. So, if the pointer arrow was spun at random and a direction was chosen, each search direction or piece number would have an equal chance to be selected. Over time, the search directions actually become weighted by the pheromone values.

In figure 4, ants would have a greater chance to select either search direction three or search direction one. For the selection process to work, a random number between one and four is generated four thousand times and is weighted by pheromones. The random number corresponds to a part of the roulette wheel as indicated in figures 3 and 4. The random number which is most often picked is selected. This number corresponds to a search direction and so this corresponding search direction becomes the actual new search direction. Essentially, the higher the pheromone concentration is in a given search direction, the more likely this search direction will be chosen again.

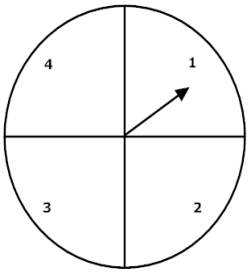

Figure 3. Initial roulette wheel

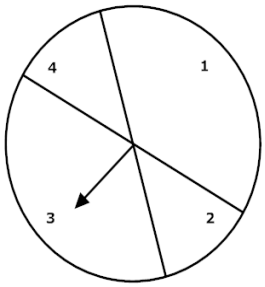

Figure 4. Weighted roulette wheel

As a result of the importance of the random generation of numbers in the selection of the search direction, care has to be taken to use a random number generator with qualities suited for this type of job. This is why the Mersenne Twister (MT) random number generator is selected to perform the random number generation. MT is used because it is very advantageous when compared to other random number generators [6]. First of all, MT has a very long period of $2^{19937}-1$. This algorithm also has a good k-distribution property and uses memory efficiently consuming only 624 words of 32 bits [26]. Another excellent quality of this randomizer is its speed, which is almost four times faster than the standard random function used in the C++ computer language [26].

3.3. Variable parameters

Many of the variable parameters used in the MCACO algorithm are based on experimentation. A wide range of different variable combinations were experimented with and values that resulted in the most accurate and stable solutions were used.

Three very important variables in the algorithm are pheromone growth rate, pheromone evaporation rate, and search radius reduction factor. When an ant follows a given search direction and finds a better fitness at a new point, pheromone needs to be added to this search direction so that new generations of ants can follow the same direction. The addition of pheromone is monitored by the pheromone growth rate. The same is true for the opposite effect when an ant finds a worse fitness along a given search direction and pheromone needs to be removed from this particular trail. This is akin to the pheromone evaporation rate. Also, over time, the search radius that ants can search in shrinks so that they can narrow down on the global minimum. The rate of shrinkage in search diameter is important, because it stipulates how fast or how slow the overall optimization process proceeds. Setting the radius reduction factor to a large value increases the number of function calls as well the accuracy of the optimized solution. The opposite is true if the radius reduction factor is set to a small value. If a normalized initial search radius set at one is used, the final minimal search radius obtained is shown in table 3.

Table 3: Final search radius values

	Number of radius reductions	Final radius value
16 initial nests	20	0.121577
3 final nests	142	0.000723

There are 20 radius reductions per nest in part one of the MCACO algorithm and 142 radius reductions per nest in part two. Many combinations of parameters were researched and the values found to perform well for the single-objective optimization test cases are shown in table 4.

Table 4: Parameters used for single-objective MCACO

Parameter Name	Value
Pheromone growth rate	1.05
Pheromone evaporation rate	0.90
Initial radius (normalized)	1.00
Radius reduction factor for Part I	0.90
Radius reduction factor for Part II	0.95

3.4. MCACO ant movement description

The general description of the single-objective MCACO algorithm ant movement developed in this thesis is detailed below:

1. Global initialization
    a. Set initial search radius
    b. Set pheromone growth and evaporation rates
    c. Set search radius reduction factor
    d. Initialize pheromone values
    e. Initialize all other variables
2. Initial trial for each nest
    a. Evaluate function at the nest
    b. Set nest fitness as current optimum in all search directions
3. Loop for each nest
    a. Choose a global search direction from the nest
    b. If this search direction is new, move the ant in the chosen search direction by a certain radius
        i. If fitness is worse than at the nest, then
            1. Update location back to nest coordinates
            2. Update global pheromone values as bad
            3. Update search radius by decreasing it

        ii. If fitness is better than at nest, then
            1. Update global pheromone values as good
            2. Update location to current coordinates
            3. Update search radius by decreasing it
            4. Update local optimum to better fitness value
    c. If search direction was previously chosen, then
        i. If location is at the nest
            1. Choose a global search direction from the nest
            2. Move an ant in the chosen direction by a certain radius
            3. If fitness is worse than at the nest, then
                a. Update location back to nest coordinates
                b. Update global pheromone values as bad
                c. Update search radius by decreasing it
            4. If fitness is better than at the nest, then

a. Update global pheromone values as good
b. Update location to current coordinates
c. Update search radius by decreasing it
d. Update local optimum to better fitness value

ii. If the location is not at the nest, then
1. Choose a local search direction
2. Move an ant in the chosen search direction by a certain radius
3. If fitness is better than at previous location, then
a. Update global pheromone values as good
b. Reset local pheromone values
c. Update location to current coordinates
d. Update search radius by decreasing it
e. Update local optimum to better fitness value
4. If fitness is worse than at the previous location, then
a. Update location by going back to previous location
b. Update global pheromone values as bad
c. Update local pheromone values as bad
d. Update search radius by decreasing it

d. Update global optimum

4. Go back to loop until maximum number of functions calls reached

3.6. NNC method description

The NNC method is used in conjunction with the MCACO algorithm in order to optimize multi-objective functions. The method is described here for the case of two-objective optimization. The two-objective optimization problem can be described as follows [23],

$$\begin{aligned} &\text{Problem P1} \\ &\min_x \{\mu_1(x) \quad \mu_2(x)\} \\ &\text{subject to:} \\ &g_j(x) \le 0, \quad (1 \le j \le r) \\ &h_k(x) = 0, \quad (1 \le k \le s) \\ &x_{li} \le x_i \le x_{ui}, \quad (1 \le i \le n_x) \end{aligned} \tag{9}$$

The functions of $\mu_1(x)$ and $\mu_2(x)$ refer to the objectives, while $g_j(x)$ and $h_k(x)$ refer to the inequality and equality constraints.

The first step in the NNC method is to solve for the anchor points. This entails splitting the multiple objective problem into two single-objective problems and solving them individually. In other words, the following problem needs to be solved [23].

$$
\begin{aligned}
&\text{Problem PU1}\\
&\min_{x} \mu_i(x), \quad (1 \le i \le n)\\
&\text{subject to:}\\
&g_j(x) \le 0, \quad (1 \le j \le r)\\
&h_k(x) = 0, \quad (1 \le k \le s)\\
&x_{li} \le x_i \le x_{ui}, \quad (1 \le i \le n_x)
\end{aligned}
\tag{10}
$$

The line connecting the two anchor points is called the utopia line [23]. The next step is to normalize the search space. Let the utopia point be defined by [23],

$$\mu^u = \left[\mu_1(x^{1*}) \qquad \mu_2(x^{2*})\right]^T \tag{11}$$

Also, let the distances between the anchor points and the utopia point be defined by [23],

$$L_1 = \mu_1(x^{2*}) - \mu_1(x^{1*}) \text{ and } L_2 = \mu_2(x^{1*}) - \mu_2(x^{2*}) \tag{12}$$

The normalized design metrics can now be evaluated as follows [23],

$$\bar{\mu} = \left[\frac{\mu_1(x) - \mu_1(x^{1*})}{L_1} \qquad \frac{\mu_2(x) - \mu_2(x^{2*})}{L_2}\right]^T \tag{13}$$

The subsequent step is to define the utopia line vector. This is the direction from the normalized utopia point one to the normalized utopia point two or as [23],

$$\bar{N}_1 = [\bar{\mu}^{2*} \qquad -\bar{\mu}^{1*}] \tag{14}$$

The following step is to compute the normalized increments along the utopia line vector which can be calculated as follows [23],

$$\delta_1 = \frac{1}{m_1 - 1} \tag{15}$$

Above, m_1 represents the number of solution points needed. The next goal is to generate the utopia line points and then evaluate that set of evenly distributed points on the utopia line as [23],

$$
\begin{aligned}
&\bar{X}_{pj} = \alpha_{1j}\bar{\mu}^{1*} + \alpha_{2j}\bar{\mu}^{2*}\\
&\text{where}\\
&0 \le \alpha_{1j} \le 1,\\
&\sum_{k=1}^{2} \alpha_{kj} = 1
\end{aligned}
\tag{16}
$$

Then, use the set of evenly distributed points generated in the previous step to obtain a set of Pareto points by solving a succession of optimization runs for problem P2 which is described as [23],

$$
\begin{aligned}
&\text{Problem P2 for } j^{th} \text{ point}\\
&\min_{x} \ \bar{\mu}_2\\
&\text{subject to:}\\
&g_j(x) \le 0, \quad (1 \le j \le r)\\
&h_k(x) = 0, \quad (1 \le k \le s)\\
&x_{li} \le x_i \le x_{ui}\\
&\bar{N}_1(\bar{\mu} - \bar{X}_{pj})^T \le 0\\
&\bar{\mu} = \left[\bar{\mu}_1(x) \quad \bar{\mu}_2(x)\right]^T
\end{aligned}
\tag{17}
$$

Each optimization, for each j point, corresponds to one Pareto point. The final step would be to use an inverse mapping which can be defined as [23],

$$\mu = \left[\bar{\mu}_1 L_1 + \mu_1(x^{1*}) \qquad \bar{\mu}_2 L_2 + \mu_2(x^{2*})\right]^T \tag{18}$$

Equation (14) gives a solution in the real function space. This useful method generates an evenly distributed set of Pareto solutions. The multi-objective version of the MCACO algorithm only uses a single nest per Pareto point optimization to decrease the overall number of function calls.

3.8. Statistical Measures

A few statistical tools are used to analyze the results obtained by the MCACO algorithm. The first measure that is used is the arithmetic mean or the average. The equation is given as follows [27],

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{19}$$

In equation (19), n denotes the sample size. The average is used to refer to a middle value. The averages of the MCACO results are supposed to closely approximate actual minimums for the algorithm to be successful.

The second statistical tool used to analyze the results is the standard deviation. The formula is given below [27]

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2} \tag{20}$$

This indicator explains roughly how far from the average the optimized solution may lie. Small standard deviations are desirable because the smaller the standard deviation the more stable the result. In the next section, classical test functions are used to measure the capacity of the algorithm.

4. Results

4.1. Single-objective optimization test cases

The results for the single-objective test cases have been obtained using the MCACO algorithm developed in this thesis. For each function, the algorithm was run 100 times, and the results and location of the optimized values were recorded. The functions tested and results obtained are given below.

Griewank function :

$$f(x,\ y) = \frac{x^2}{4000} + \frac{y^2}{4000} - \cos\left(\frac{x}{\sqrt{1}}\right)\cos\left(\frac{y}{\sqrt{2}}\right) + 1$$

for $x \in [-10,\ 10]$ and $y \in [-10,\ 10]$

The global minimum is $f(x,\ y) = 0$ located at $(x,\ y) = (0,\ 0)$

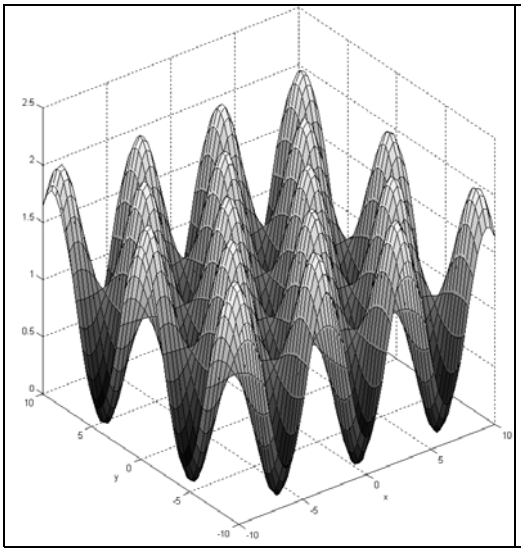

Figure 5. Topology of the Griewank's function

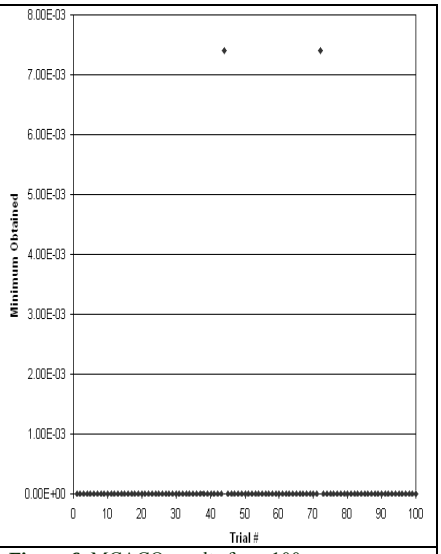

Figure 6. MCACO results from 100 runs

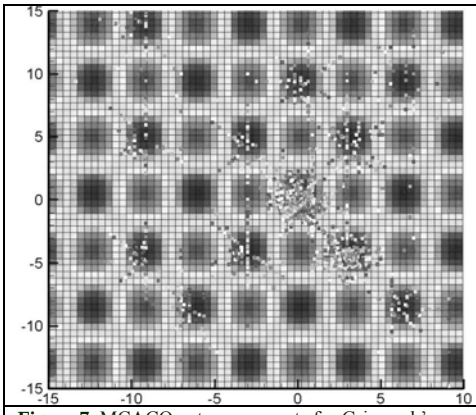

Figure 7. MCACO ants movements for Griewank's function

	Average	Standard deviation
Minimum obtained	0.0001481	0.0010407
X location	0.0627627	0.4418653
Y location	0.0001186	0.6308250

Table 5. MCACO results for the Griewank's function

Overall, the accuracy results for the single-objective cases are very good (table 6). Almost every function has a small value of standard deviation for the solution, x value, and y value. This signifies that the results are very stable over all of the test runs. However, there are a few cases for several functions where a local minimum was found as opposed to the global minimum. This happened, for example, in the case of the Rosen function.

Table 6: Comparison of minima obtained using MCACO for single-objective test functions

Test Function Name	Actual minimum	Averaged MCACO minimum
Beale	0	0.0043947
Bohachevsky	0	0.0003941
Booth	0	0.0000113
Branin	0.397887	0.3978878
Easom	-1	-0.9996689
Goldstein and Price	3	3.0000918
Freudenstein and Roth	0	0.0390358
Hump	0	0.0000017
Griewank	0	0.0001481
Matyas	0	0.0000377
Michalewics	-1.8013	-1.7945548
Rastrigin	0	0.0000656
Rosenbrock	0	0.0617681
Martin and Gaddy	0	0.0000048
Shubert	-186.7309	-186.7302400

Rosen	-18.5680	-18.1634670
Ackley	0	0.0016163
Perm #1	0	0.5796424
Perm #2	0	0.4264297
Sphere	0	0.0000003

The number of function calls for all single-objective functions was three thousand function calls.

The MCACO results are better than the results for the original CACO and for CIAC. MCACO is also roughly on the same level as the results for binary ant system. However, when compared to DACO and ACO_R, MCACO uses many more function calls. Although it is not state of the art, MCACO compares relatively well to some of the other continuous ant algorithms.

The algorithm with the least number of function calls, and hence the algorithm with the best result, is ACO_R. This algorithm has a very strong connection to the original ACO algorithm because it also performs an incremental construction of solutions [14]. The fundamental idea in ACO_R is the shift from using a discrete probability distribution to using a continuous one. A continuous probability distribution can be modeled by using a probability density function (PDF). ACO_R uses a Gaussian kernel, which is a weighted sum of several one-dimensional Gaussian functions. This kernel is denoted as follows [14]:

$$G^i(x) = \sum_{l=1}^{k} \omega_l g_l^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} e^{-\frac{(x-\mu_l^i)^2}{2\sigma_l^{i2}}} \quad (21)$$

In ACO_R, the ants sample a PDF, such as equation (21), to construct solutions. In fact, the ACO metahueristic is in a way similar to the ACO_R solution procedure. Based on the explanation above, ACO_R logically differs from MCACO.

However, MCACO does not compare well to other optimization algorithms when total number of function evaluations is counted since it requires significantly more function evaluations. This is because ant colony routines were first created for discrete optimization problems and later extended to continuous ones as opposed to other schemes which were initially created for continuous functions.

4.2. Multi-objective optimization test cases

The multi-objective results are very good for the functions tested. The number of function calls is ten thousand one hundred for each function. The multi-objective results were obtained with the help of the NNC method and so a small amount of work is required

before the functions can be handled by MCACO. For each function, the routine was run 50 different times and the results for the best case were recorded. The results are given in the figures that follow:

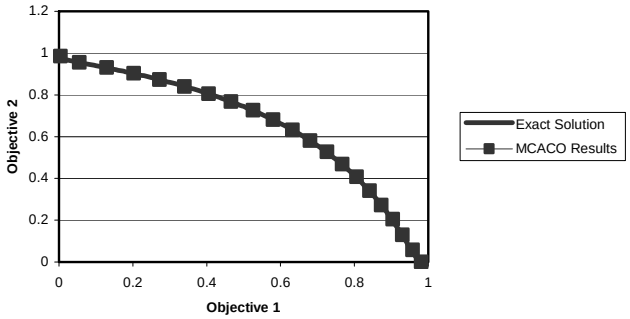

Figure 8. Fonseca and Flemming two-objective test function optimization results

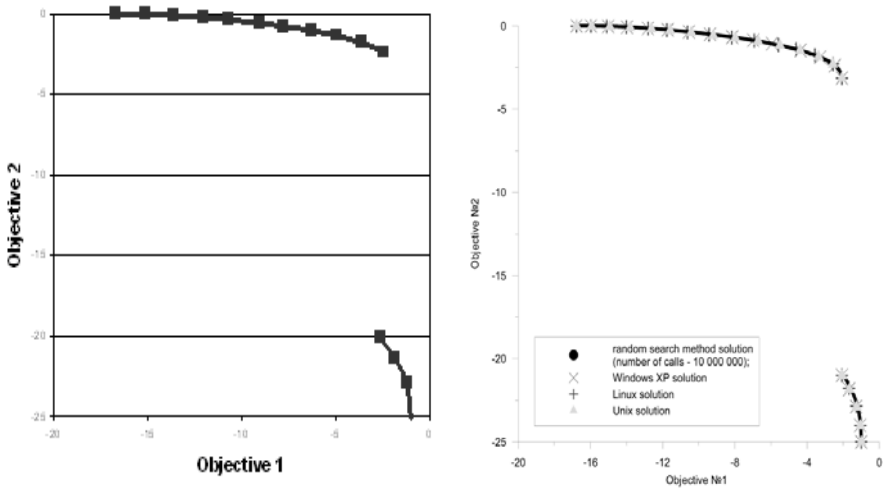

Figure 9. Poloni two-objective test function; results from MCACO and those obtained with IOSO software [28]

5. Recommendations for Future Research

A Pareto filter can be built into the MCACO algorithm when it is used with the NNC method to optimize multi-objective problems. This would take away the need to check the solutions afterward to see if they are Pareto optimal. By changing certain values, such as the radius reduction factor, the number of function calls can be lowered or raised to decrease or increase the accuracy. It would also be prudent to try using a different number of nests to see how the number of function calls, the stability, and the accuracy of the algorithm is affected. Two more ideas that can be explored include adding constraint

handling to the MCACO code and investigating functions that have the global minimum located exactly on the domain boundary [29]. The number of function calls and the success rates for some of the easier functions are excellent. However, when the more difficult optimization test cases are tried, MCACO becomes unstable and the success rate drops.

The last recommendation for future research is to fit a local response surface for each nest. This technique has the capacity to drastically reduce the amount of function calls.

In summary, the MCACO algorithm was developed based on the principles of the CACO algorithm and by using the underlying ideas of ACO. The MCACO algorithm was tested for single-objective optimization problems and, in conjunction with the NNC method, was tested for multi-objective optimization problems. The results obtained using the MCACO routine indicates that the method is stable and accurate. The method is deemed stable because the standard deviation for all important values, such as the averaged global minima obtained and their respective locations, is very small. Accuracy is very good for the MCACO method because the MCACO results indicate that the minima obtained for the test cases closely resemble the true analytic minima. Although it is not yet comparable with other top-tier optimization methods in terms of function calls, there is still room for improvement.

References

[1] M. Dorigo (1992) Optimization, Learning and Natural Algorithms, Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano.

[2] M. Dorigo and T. Stützle (2004) *Ant Colony Optimization,* London, The MIT Press.

[3] M. Dorigo, M. Birtattari, and T. Stützle (2006) Ant Colony Optimization, Artificial Ants as a Computational Intelligence Technique, Technical Report No. TR/IRIDIA/2006-023, Universite Libre de Bruxelles.

[4] G. Bilchev and I. Parmee (1995) The Ant Colony Metaphor for Searching Continuous Design Spaces, *Proceedings of the AISB Workshop on Evolutionary Optimization*, Berlin, pp. 25-39.

[5] L. Kuhn (2002) Ant Colony Optimization for Continuous Spaces, thesis, Department of Information Technology and Electrical Engineering, University of Queensland, Australia.

[6] M. Matsumoto and T. Nishimura (1998) Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator, *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3-30.

[7] E. Chong and S. Zak (2001) *An Introduction to Optimization, 2nd Edition,* New York, Wiley-Interscience.

[8] W. Sun and Y. Yuan (2006) *Optimization Theory and Methods: Nonlinear Programming*, New York, Springer.

[9] M. Dorigo and K. Socha (2006) An Introduction to Ant Colony Optimization, Technical Report No. TR/IRIDIA/2006-010, Universite Libre de Bruxelles.

[10] E. Bonabeau, M. Dorigo, and G. Theraulaz (1999) *Swarm Intelligence: From Natural to Artificial Systems,* New York, Oxford University Press.

[15] M. Mathur, S. Karale, S. Priye, V. Jayaraman, and B. Kulkarni (2000) Ant Colony Approach to Continuous Function Optimization, *Industrial & Engineering Chemistry Research*, vol. 39, pp. 3814-3822.

[11] M. Dorigo and K. Socha (2008) Ant colony optimization for continuous domains, *European Journal of Operational Research*, vol. 158, no. 3, pp. 1155-1173.
[12] G. Bilchev and I. Parmee (1996) Constrained Optimization With An Ant Colony Search Model, *Proceedings of ACEDC'96*, Plymouth, UK, pp. 141-151.
[13] J. Dreo and P. Siarry (2004) Continuous interacting ant colony algorithm based on dense heterarchy, *Future Generation Computer Systems*, vol. 20, no. 5, pp. 841-856.
[14] M. Kong and P. Tian (2006) A Direct Application of Ant Colony Optimization to Function Optimization Problem in Continuous Domain, *Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006, Proceedings*, pp. 324-331.
[16] K. Deb (2001) *Multi-Objective Optimization Using Evolutionary Algorithms*, New York, Wiley.
[17] R. Marler and J. Arora (2004) Survey of Multi-Objective Optimization Methods for Engineering, *Structural and Multidisciplinary Optimization,* vol. 26, no. 6, pp. 369-395.
[18] K. Miettinen (1998) *Nonlinear Multiobjective Optimization*, New York, Springer.
[19] Y. Donoso and R. Fabregat (2007) *Multi-Objective Optimization in Computer Networks Using Metaheuristics*, Chicago, Auerbach.
[20] I. Das and J. Dennis (1997) A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems, *Structural and Multidisciplinary Optimization,* vol. 14, no. 1, pp. 63-69.
[21] A. Messac, G. Sundararaj, R. Tappeta, and J. Renaud (2000) Ability of Objective Functions to Generate Points on Non-Convex Pareto Frontiers, *AIAA Journal*, Vol. 38, No. 6, pp. 1084-1091.
[22] A. Messac and C. Mattson (2004) Normal Constraint Method with Guarantee of Even Representation of Complete Pareto Frontier, *AIAA Journal*, Vol. 42, No. 10, pp. 2101-2111.
[23] A. Messac, A. Ismail-Yahaya, and C. Mattson (2003) The Normalized Normal Constraint Method for Generating the Pareto Frontier, *Structural and Multidisciplinary Optimization*, Vol. 25, No. 2, pp. 86-98.
[24] M. Martinez, J. Sanchis, and X. Blasco (2007) Global and Well-Distributed Pareto Frontier by Modified Normalized Normal Constraint Methods for Bicriterion Problems, *Structural and Multidisciplinary Optimization*, Vol. 34, No. 3, pp. 197-209.
[25] J. Sanchis, M. Martinez, X. Blasco, and J. Salcedo (2007) A New Perspective on Multiobjective Optimization by Enhanced Normalized Normal Constraint Method, *Structural and Multidisciplinary Optimization*, Springer Berlin.
[26] M. Matsumoto and T. Nishimura (2000) Dynamic Creation of Pseudorandom Number Generators, *Monte Carlo and Quasi-Monte Carlo Methods 1998*, Springer, pp. 56-69.
[27] J. Kapur and H. Saxena (1997) *Mathematical Statistics,* New Delhi, S. Chand.
[28] I. Egorov (2003) IOSO NM Version 1, User Guide, IOSO Technology Center.
[29] M. Kong and P. Tian (2005) A Binary Ant Colony Optimization for the Unconstrained Function Optimization Problem, *Lecture Notes in Computer Science*, Vol. 3801, pp. 682-687.