Florida International University
FIU Digital Commons

FIU Electronic Theses and Dissertations | University Graduate School

7-15-2008

Modified continuous ant colony algorithm for function optimization

Alexandre Aidov
Florida International University

Follow this and additional works at: http://digitalcommons.fiu.edu/etd

 Part of the Mechanical Engineering Commons

Recommended Citation
Aidov, Alexandre, "Modified continuous ant colony algorithm for function optimization" (2008). *FIU Electronic Theses and Dissertations.* Paper 1166.
http://digitalcommons.fiu.edu/etd/1166

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

MODIFIED CONTINUOUS ANT COLONY ALGORITHM FOR FUNCTION OPTIMIZATION

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

MECHANICAL ENGINEERING

by

Alexandre Aidov

2008

To: Interim Dean Amir Mirmiran
College of Engineering and Computing

This thesis, written by Alexandre Aidov, and entitled Modified Continuous Ant Colony Algorithm for Function Optimization, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Yiding Cao

Igor Tsukanov

George S. Dulikravich, Major Professor

Date of Defense: July 15, 2008

The thesis of Alexandre Aidov is approved.

Interim Dean Amir Mirmiran
College of Engineering and Computing

Dean George Walker
University Graduate School

Florida International University, 2008

DEDICATION

I dedicate this thesis to my grandmother. Without her support and nurturing, this work would have never been possible. I also dedicate this work to my parents for their sacrifice and to my fiancé for her patience and love.

ACKNOWLEDGMENTS

I wish to thank members of my committee for their support, patience, and intellectual discussions. Professor Yiding Cao was particularly helpful in arming me with the necessary quantitative procedures I needed. Professor Igor Tsukanov was helpful in providing me with the confidence I needed to successfully complete this project. I would like to thank my major professor, Professor George S. Dulikravich. From the initial steps, he was influential in guiding me through my research. He also provided me with necessary ideas to continue forward and make my ant colonies move. I also want to thank Professor Dulikravich for the financial support and the intellectual freedom he gave me to pursue this work in my own way.

I also would like to thank my fellow researchers, Ramon Moral and Souma Chowdhury, for all the ideas and help they gave me.

ABSTRACT OF THE THESIS

MODIFIED CONTINUOUS ANT COLONY ALGORITHM FOR FUNCTION OPTIMIZATION

by

Alexandre Aidov

Florida International University, 2008

Miami, Florida

Professor George S. Dulikravich, Major Professor

Many classical as well as modern optimization techniques exist. One such modern method belonging to the field of swarm intelligence is termed ant colony optimization. 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 thesis, a novel ant colony optimization technique for continuous domains was developed. The goal was to provide improvements in computing time and robustness when compared to other optimization algorithms. Optimization function spaces can have extreme topologies and are therefore difficult to optimize. The proposed method effectively searched the domain and solved difficult single-objective optimization problems. The developed 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.

TABLE OF CONTENTS

CHAPTER		PAGE
1.	INTRODUCTION	1
2.	LITERATURE REVIEW AND THEORY	5
	2.1. Optimization	5
	2.2. Single-Objective Optimization	7
	2.3. Combinatorial Optimization	8
	2.4. Ant Colony Optimization	10
	2.5. ACO Algorithms for Continuous Domains	14
	2.6. Continuous Ant Colony Optimization	15
	2.7. Multi-Objective Optimization	17
	2.8. Methods for Multi-Objective Optimization	19
	2.9. Normalized Normal Constraint Method	22
3.	METHODOLOGY	24
	3.1. Modified Continuous Ant Colony Optimization	24
	3.2. Ant and Nest Movement	25
	3.3. Search Direction Selection	27
	3.4. Variable Parameters	29
	3.5. MCACO Ant Movement Description	31
	3.6. MCACO Functions and Function Calls	34
	3.7. NNC Method Description	37
	3.8. Statistical Measures	39
4.	MCACO RESULTS	41
	4.1. Single-Objective Optimization Test Cases	41
	4.2. Multi-Objective Optimization Test Cases	62
5.	DISCUSSION	66
	5.1. Results and Comparisons	66
	5.2. Benefits and Advantages	73
	5.3. Difficulties and Limitations	75
6.	CONCLUSION	80
	6.1. Recommendations for Future Research	80
	6.2. Summary	82
LIST OF REFERENCES		83
APPENDICES		86

LIST OF TABLES

TABLE PAGE

1. ACO concepts....11

2. Applications of ACO....13

3. Continuous ant based optimization techniques....14

4. Weighted global criterion methods....20

5. Final search radius values....30

6. Parameters used for single objective MCACO....31

7. Beale function optimization with MCACO....42

8. Bohachevsky function optimization with MCACO....43

9. Booth function optimization with MCACO....44

10. Branin function optimization with MCACO....45

11. Easom function optimization with MCACO....46

12. Goldstein and Price function optimization with MCACO....47

13. Freudenstein and Roth function optimization with MCACO....48

14. Hump function optimization with MCACO....49

15. Griewank function optimization with MCACO....50

16. Matyas function optimization with MCACO....51

17. Michalewics function optimization with MCACO....52

18. Rastrigin function optimization with MCACO....53

19. Rosenbrock function optimization with MCACO....54

20. Martin and Gaddy function optimization with MCACO....55

21. Shubert function optimization with MCACO....56

22. Rosen function optimization with MCACO.....57

23. Ackley function optimization with MCACO.....58

24. Perm #1 function optimization with MCACO.....59

25. Perm #2 function optimization with MCACO.....60

26. Sphere function optimization with MCACO.....61

27. Comparison of minimums obtained using MCACO.....67

28. Ant colony based algorithm comparison.....68

29. Non-ant based algorithm comparison.....70

30. Previous version MCACO results.....81

LIST OF FIGURES

FIGURE PAGE

1. Easom function geometry....3

2. Optimization approaches....5

3. Plot of $f(x) = x^2 - 2$....6

4. Optimization classification based on objectives....6

5. Optimization classification based on function space....8

6. Non-optimized TSP....9

7. Optimized TSP....9

8. Pheromone explanation....10

9. CACO nest with eight search directions....15

10. Pareto optimal points....19

11. Concave Pareto curve....21

12. Uneven Pareto point spread....21

13. MCACO algorithm....24

14. Set one search directions....26

15. Set two search directions....26

16. Random search directions....27

17. Initial roulette wheel....28

18. Weighted roulette wheel....28

19. Beale function optimization results....41

20. Beale function....41

21. Bohachevsky function optimization results....42

22. Bohachevsky function....42
23. Booth function optimization results....43
24. Booth function....43
25. Branin function optimization results....44
26. Branin function....44
27. Easom function optimization results....46
28. Easom function....46
29. GP function optimization results....47
30. GP function....47
31. FR function optimization results....48
32. FR function....48
33. Hump function optimization results....49
34. Hump function....49
35. Griewank function optimization results....50
36. Griewank function....50
37. Matyas function optimization results....51
38. Matyas function....51
39. Michalewics function optimization results....52
40. Michalewics function....52
41. Rastrigin function optimization results....53
42. Rastrigin function....53
43. Rosenbrock function optimization results....54

44. Rosenbrock function....54
45. MG function optimization results....55
46. MG function....55
47. Shubert function optimization results....56
48. Shubert function....56
49. Rosen function optimization results....57
50. Rosen function....57
51. Ackley function optimization results....58
52. Ackley function....58
53. Perm #1 function optimization results....59
54. Perm #1 function....59
55. Perm #2 function optimization results....60
56. Perm #2 function....60
57. Sphere function optimization results....61
58. Sphere function....61
59. Fonseca and Flemming function optimization results with MCACO....62
60. Poloni function optimization results with MCACO....63
61. Binh function optimization results with MCACO....64
62. Lis function optimization results with MCACO....65
63. Rendon function optimization results with MCACO....65
64. MCACO Fonseca and Flemming comparison with exact solution....71
65. MCACO Poloni comparison with IOSO solution....71
66. MCACO Binh comparison with exact solution....72

67. MCACO Lis comparison with IOSO solution....72
68. MCACO Rendon comparison with IOSO solution....73
69. Branin function ants and minimums....74
70. Rosen function in xy plane....76
71. Rendon function xz view....77
72. Rendon function yz view....77
73. Booth function ant movement....86
74. Goldstein and Price function ant movement....87
75. Griewank function ant movement....87
76. Rastrigin function ant movement....88
77. Beale initial ant placement....89
78. Bohachevsky initial ant placement....89
79. Booth initial ant placement....89
80. Branin initial ant placement....89
81. Easom initial ant placement....89
82. GP initial ant placement....89
83. FR initial ant placement....90
84. Hump initial ant placement....90
85. Griewank initial ant placement....90
86. Matyas initial ant placement....90
87. Michalewics initial ant placement....90
88. Rastrigin initial ant placement....90

89. Rosenbrock initial ant placement....91

90. MG initial ant placement....91

91. Shubert initial ant placement....91

92. Rosen initial ant placement....91

93. Ackley initial ant placement....91

94. Perm # 1 initial ant placement....91

95. Perm #2 initial ant placement....92

96. Sphere initial ant placement....92

CHAPTER 1

INTRODUCTION

Optimization is an important aspect of numerous scientific endeavors, be it involving natural sciences, social sciences, or engineering. There are two types of optimization problems. The first type has a single objective and the second type of problem 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]. Although the groundwork for CACO has already been laid, there is room for improvement. The abovementioned optimization technique belongs to the category of swarm intelligence. The CACO algorithm tries to mimic the foraging behavior of ants in order to transform this behavior into a viable optimization approach. It might seem that ants behaviorally are unsophisticated little critters, but in fact, when working together they can perform complicated tasks such as optimizing the search for food [5].

Many optimization methods in existence are able to optimize continuous functions; CACO falls into this category. The goal of this research is to create and modify a CACO algorithm that 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).

Real life engineering optimization problems are complicated and require a lot of computing time. More computing time equates to higher cost. In effect, it is important for any optimization scheme to keep the cost, or computing time, as small as possible.

There exists an abundance of classical optimization test problems and real life problems that are very different from each other. Optimization problems in function form can have very diverse topologies. The Easom function, for example, which has one sharp minimum over the whole domain, comes to mind.

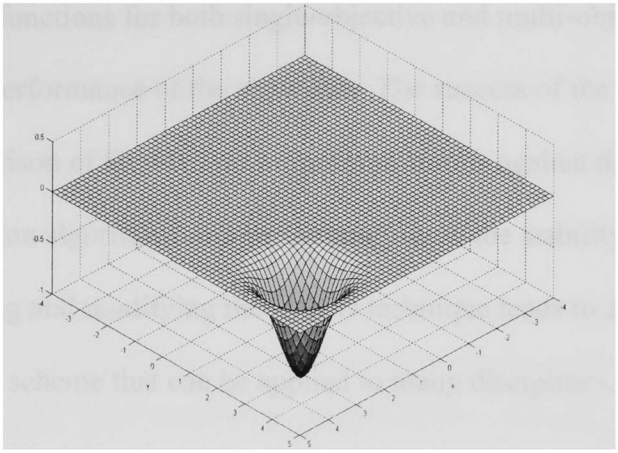


Figure 1: Easom function geometry

Many optimization algorithms perform poorly on special functions such as the one described above. The MCACO algorithm circumvents such discrepancies in performance and increases stability of the search process for the global minimum in such irregular functions. 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 direction, ant movement, fitness evaluation, 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. The success of the method can be gauged by the comparison of the MCACO algorithm results against the results obtained using other optimization algorithms and by the analysis of the stability and robustness of the method. Expanding and modifying the CACO technique leads to an improved function optimization scheme that can be applied to many disciplines.

CHAPTER 2

LITERATURE REVIEW AND THEORY

2.1. Optimization

Optimization is the branch of mathematics which involves the quantitative study of optima and the methods for finding them [7]. In other words, optimization can be described as obtaining the "best" solution to an optimization design problem. Optimization problems are encountered in many disciplines including engineering, economics, mathematics, and physics. The four general approaches to optimization are given in the following figure [7]:

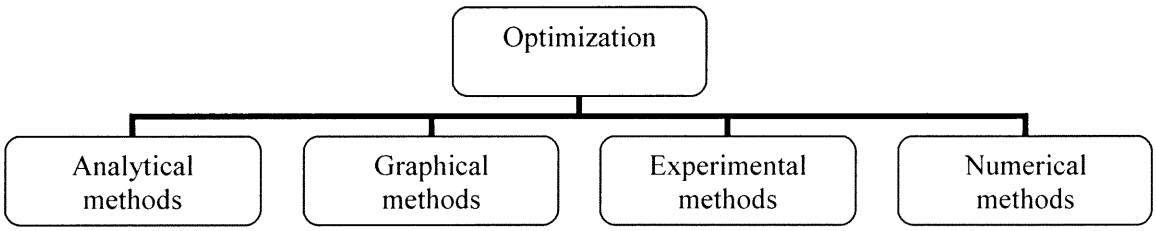


Figure 2: Optimization approaches

Analytical methods are based on the techniques of differential calculus and on the calculus of variations. For example, a function can be differentiated and the zeroes can be located as shown below,

$$\begin{aligned} &f(x) = x^3 + 3x^2 - 24x + 3 \\ &f'(x) = 3x^2 + 6x - 24 \\ &f'(x) = 0 \quad \text{at} \quad x_1 = 2 \quad \text{and} \quad x_2 = -4 \end{aligned} \tag{1}$$

In this case, x_1 is a local minimum and x_2 is a local maximum. Graphical methods involve plotting functions and visually discerning where the optimum is located. For example, take into consideration, the following function,

$$\begin{aligned} &f(x) = x^2 - 2 \\ &x \in [-2,\ 2] \end{aligned} \tag{2}$$

The function of equation (2) can be plotted in the Cartesian coordinate system as follows:

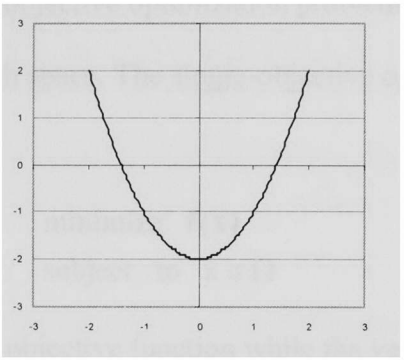


Figure 3: Plot of $f(x) = x^2 - 2$

By inspection of the plot, the minimum is located at (0,-2). Experimental methods involve direct experimentation on a system to achieve optimum performance. For example, if designing a car for maximum speed, different versions of the car can be built and tested for optimum speed. Numerical methods are computational techniques that are able to solve highly complex optimization problems. A few examples of numerical optimization techniques include simulated annealing and genetic algorithms. Optimum seeking methods are also known as mathematical programming techniques [8].

Optimization can, alternatively, fall into the categories shown in the following figure:

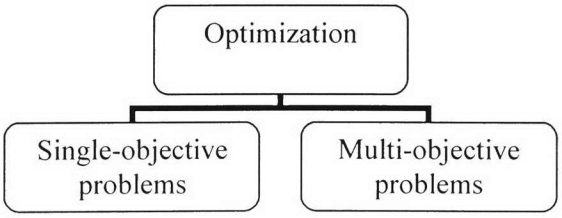


Figure 4: Optimization classification based on objectives

The simplest type of optimization problem is the single-objective optimization problem. This specific optimization problem will be explained in the next section.

2.2. 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 [9],

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad x \in \Omega \end{aligned} \tag{3}$$

The function f(x) is called the objective function while the vector x is a vector of n independent variables denoted by $x = [x_1, x_2, ..., x_n]^T$. 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 [9]. 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 [10],

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad x \in \Omega \\ &c_i(x) = 0, \quad i \in E \\ &c_i(x) \geq 0, \quad i \in I \end{aligned} \tag{4}$$

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 [10]. The reason why equations (3) and (4) are formulated as minimization problems is because minimizing a function $f(x)$ is equivalent to maximizing $-f(x)$ [9]. As a general rule, optimization problems are usually defined as minimization problems. This standard will be followed throughout the rest of the thesis.

It is known that another taxonomy of optimization problems exists and is given in the following figure:

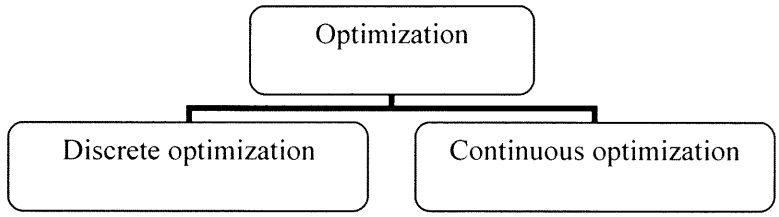


Figure 5: Optimization classification based on function space

Discrete optimization will be discussed in the next section.

2.3. 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 [11],

$$\begin{aligned} &\text{A model } P = (S, \Omega, f) \\ &\text{A search space S defined over a finite set of discrete variables} \\ &\text{A set } \Omega \text{ of constraints among variables} \\ &\text{An objective function f to be minimized} \end{aligned} \tag{5}$$

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. The figure that follows shows a

sample graph of the TSP with a given solution of routes that a salesman might take:

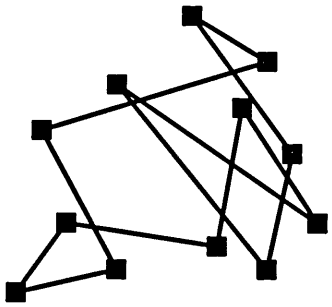


Figure 6: Non-optimized TSP

The nodes in the figure denote the cities and the lines connecting them denote the possible routes. The following figure will show the optimized solution to the sample TSP problem:

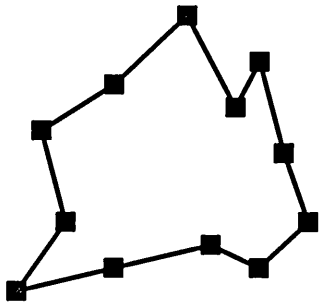


Figure 7: Optimized TSP

The optimized solution provides the salesman with the shortest possible overall trip to visit all of the cities. The idea of combinatorial optimization is very important in this study because the ACO algorithm is formulated to solve combinatorial optimization problems.

2.4. Ant Colony Optimization

ACO belongs to the optimization field of swarm intelligence. This field involves optimization algorithms inspired by the collective, natural behavior of large groups of the same species such as bees, ants, fish, and birds. ACO takes inspiration from the foraging behavior of real ants. Essentially, ACO is a probabilistic technique for solving computational optimization problems which can be reduced to finding good paths through construction graphs [2].

One of the most important topics in ACO theory is the concept of stigmergy. Stigmergy 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 [12]. 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 following figure:

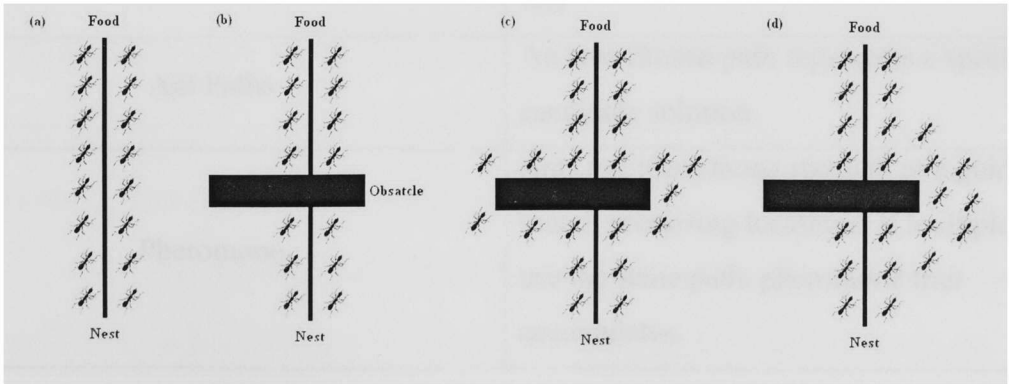


Figure 8: Pheromone explanation

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 [11]. Over time, the rest of the ants follow the eastern path between the nest and the food source as shown in part (d).

In the ACO algorithm, pheromone trails are paths laid with pheromones by the ants. 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]. The basic ACO concepts are given in the following table [2]:

Table 1: ACO concepts

Concept	Explanation
Ant movement	Ants move between nodes on a graph. Ants move by applying a probabilistic decision rule
Ant Paths	An ants chosen path represents a specific candidate solution
Pheromone	Ants use pheromone strength as a guide to search promising locations. If multiple ants use the same path, pheromone trail accumulates.

As previously stated, ACO was initially formulated as a metahueristic for combinatorial optimization problems. A metahueristic is a set of algorithmic concepts that can in turn be used to define solution methods applicable to many different problem types [2]. The ant colony optimization metahueristic pseudo-code is given in the algorithm that follows [11]:

```
Set parameters, initialize pheromone trails
While termination conditions not met do
    ConstructAntSolutions
    ApplyLocalSearch
    UpdatePheromones
End while
```
(6)

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 [11].

Using the ACO algorithm in combination with the ideas of stigmergy, many interesting problems can be solved. The following table portrays a few of the applications that have been solved using ACO [3]:

Table 2: Applications of ACO

Routing type problems	Assignment type problems	Scheduling type problems	Subset type problems	Other problems
Traveling salesman	Quadratic assignment	Project scheduling	Set covering	Classification rules
Vehicle routing	Course timetabling	Total weighted tardiness	Multiple knapsack	Bayesian networks
Sequential ordering	Graph coloring	Open shop	Maximum clique	Protein folding

Take note that all of the applications shown in the table above are problems of a discrete nature [3].

Since the ACO metahueristic was first introduced, it has gone through a number of variations to try to improve it. The first ACO routine was called Ant System (AS) [12]. The two main phases of the AS algorithm include the ants' solution construction and the pheromone update. Over time, several variants of and improvements to the ACO technique were developed. The first variant is called Ant Colony System (ACS). It uses a different transition rule and a different pheromone trail update rule. ACS also introduces the notion of local updates of pheromones and the candidate list [12]. Another alteration to the original formula is called the Max-Min Ant System (MMAS). The changes that were made include allowing only the best ants to update pheromone trails, restricting pheromone trail values to a specified interval, and initializing trails to their maximum value [12]. A few more successors to the original formulation of ACO include Elitist Ant

System, Ant-Q, and Rank-Based Ant System[2]. 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, in the current state described, the ACO algorithm cannot be used to optimize continuous optimization problems because the algorithm is only prescribed for discrete optimization problems.

2.5. ACO Algorithms for Continuous Domains

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

Table 3: Continuous ant based optimization techniques

Method name	Reference	Abbreviation
Continuous Ant Colony Optimization	[15]	CACO
Continuous Interacting Ant Colony	[16]	CIAC
Direct Ant Colony Optimization	[17]	DACO
ACO extended to continuous domains	[14]	ACO_R

CACO is based on a local search in the vicinity of a nest [15]. 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 [16]. 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 [17]. 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.

2.6. Continuous Ant Colony Optimization

CACO was the first ant colony based technique developed that was suitable for continuous function optimization [13]. 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].

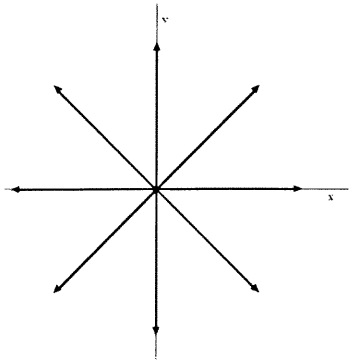


Figure 9: CACO nest with eight search directions

Figure 9 shows the nest concept pictorially with eight initial search direction vectors. 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                                          (7)
      add_trail  A(t)
      send_ants  A(t)
      evaluate  A(t)
      evaporate  A(t)
   end
end
```

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 [15].

When a chosen search direction does not result in improvement, 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 lifted 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 [13].

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.

2.7. 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 [18]. The reasons explained above clarify why single-objective optimization algorithms do not work on multi-objective optimization problems.

The general multi-objective optimization problem can be stated in the following form [19],

$$\begin{aligned} \underset{x}{\text{Minimize}} \quad & F(x) = \left[F_1(x), F_2(x), ..., F_k(x)\right]^T \\ \text{subject to} \quad & g_j(x) \le 0, \quad j = 1, 2, ..., m \\ & h_l(x) = 0, \quad l = 1, 2, ..., e \\ & x_l \le x \le x_u \end{aligned} \tag{8}$$

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 [19].

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 [20],

Assume S is feasible region,
A decision vector $x^* \in S$ is Pareto
optimal if there does not $\exists$
another decision vector $x \in S$ such that
$f_i(x) \leq f_i(x^*)$ for $\forall$ i and
$f_j(x) < f_j(x^*)$ for at least one j (9)

There are theoretically an infinite number of Pareto optimal solutions for every multi-objective optimization problem [20]. The following figure shows the Pareto optimal points of a given set of points:

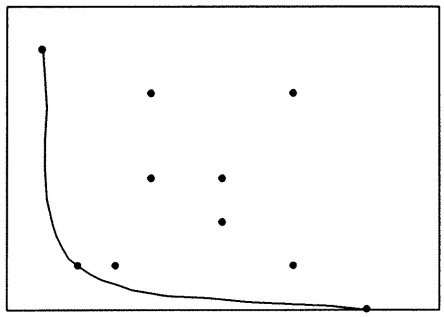


Figure 10: Pareto optimal points

The three Pareto optimal points in figure 10 are connected by a line. With the use of techniques designed specifically to solve multi-objective optimization problems, many of these Pareto points can be obtained.

2.8. 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 [21]. 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 [21]. 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. The table below shows three of the most popular weighted methods [19]:

Table 4: 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	$U = \prod_{i=1}^{k} [F_i(x)]^{w_i}$

In the table above, U represents final combined single-objective function and w_i represents the weights used. The three methods differ in the way that they build up the single-objective function [19].

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 convex Pareto curves. A concave Pareto curve is shown in the following figure:

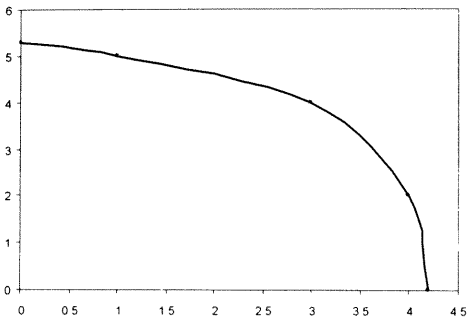


Figure 11: Concave Pareto curve

If the Pareto curve 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 [22]. This deficiency is shown in the following figure:

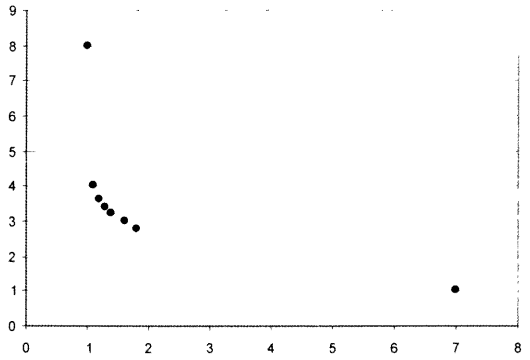


Figure 12: Uneven Pareto point spread

The majority of Pareto optimal points are grouped together in the middle and thus do not produce a good spread of solutions along the Pareto frontier.

A solution to the curvature deficiency would be to use the weighted compromise method [new 23]. The formulation of the weighted compromise method is given as follows [new 23],

$$\text{minimize } f(x)=\sum_{i=1}^{m} w_i \left(f_i(x)\right)^{c_i} \qquad (10)$$

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. As a result, 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, is called the Normalized Normal Constraint method [24].

2.9. Normalized Normal Constraint Method

The Normalized Normal Constraint (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 [24]. 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 curve [24].

Under certain uncommon circumstances, the NNC method can generate non-Pareto and weak Pareto solutions [25]. When the aforementioned occurs, a Pareto filter can be used [25]. It is an algorithm that eliminates all dominated points from the solution set [26]. To avoid another pitfall related to scaling deficiencies, the optimization is performed in the normalized objective space [27]. Having addressed the issues at hand, the methods behind the MCACO algorithm can now be explained.

CHAPTER 3

METHODOLOGY

3.1 Modified Continuous Ant Colony Optimization

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.

MCACO is very versatile algorithm. Many of the variables contained in it are user defined and can be altered if necessary. MCACO can be tailored to suit different types of problems.

The single-objective version of the MCACO algorithm can be broken up into two main parts as shown in the figure that follows:

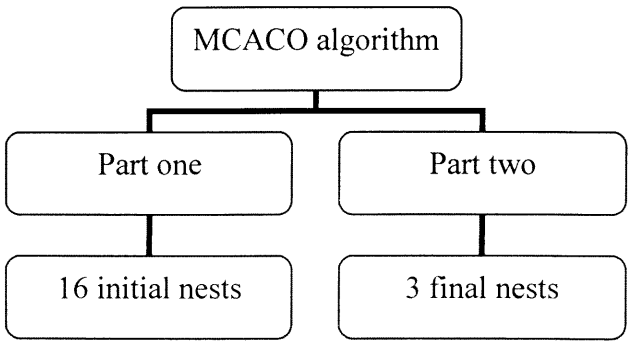


Figure 13: MCACO algorithm

Part one of the algorithm features sixteen initial nests spread across the search domain. Refer to the appendix for actual locations of initial nests. 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.

3.2. 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. The following figure shows these search directions:

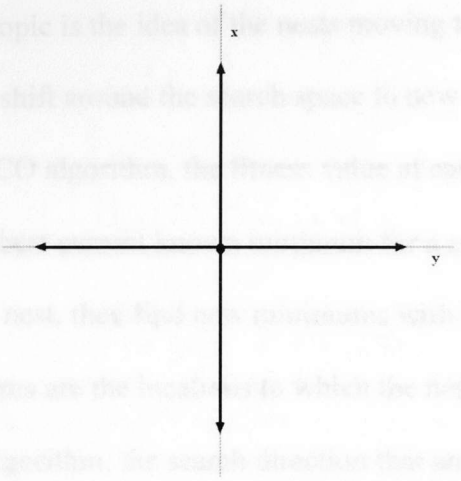


Figure 14: Set one search directions

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.

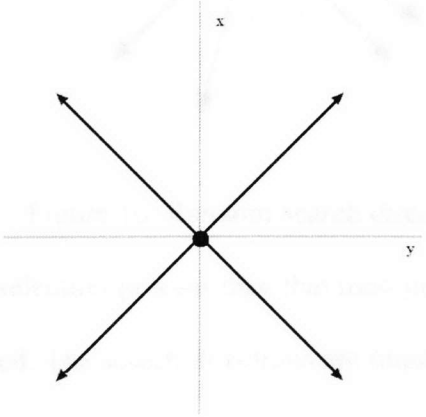


Figure 15: Set two search directions

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. But as the ants explore out from each nest, they find new minimums with lower values of function fitness. These new minimums 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.

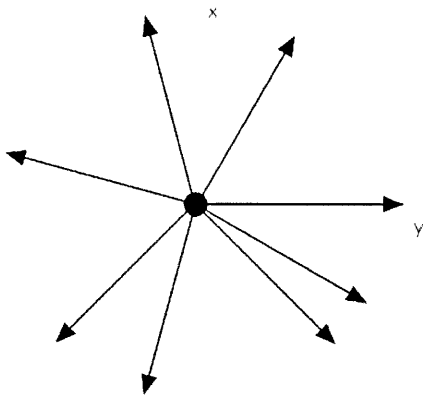


Figure 16: Random search directions

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.

3.3. Search Direction Selection

In the MCACO algorithm, the search direction is selected through the roulette wheel concept. This concept is explained in the figure that follows:

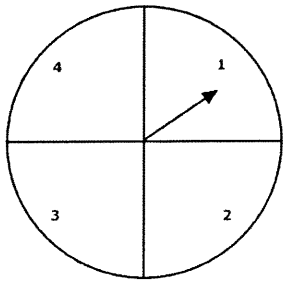


Figure 17: Initial roulette wheel

Assume each of the numbered pieces of the wheel to represent one of the possible four search directions. Initially, as shown in figure 17, 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 as shown in the figure that follows:

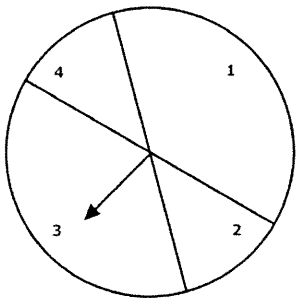


Figure 18: Weighted roulette wheel

In figure 18, 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 17 and 18. 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.

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 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.

Thus, search directions are chosen meticulously and with the help of the Mersenne Twister pseudorandom number generating algorithm. 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 [28]. 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 [28]. This particular algorithm is used because it can choose a random number within a given range very efficiently and with no serial correlation.

3.4. 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 the following table:

Table 5: 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 the following table:

Table 6: Parameters used for single-objective MCACO

Parameter Name	Value
Pheromone growth rate	1.05
Pheromone evaporation rate	0.90
Initial radius (normalized)	1.0
Radius reduction factor for part one	0.90
Radius reduction factor for part two	0.95

3.5. 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. MCACO Functions and Function Calls

Functions play a vital role in the MCACO algorithm. Many of the techniques regulated to different tasks are separated into different functions. For example, local pheromone updates and global pheromone updates are two different functions. Global pheromones are attached to the nest and are applied to directions leading out from the nest. Local pheromones are pheromones attached to the subsequent nests that are created once an ant finds a better fitness value in a certain direction. Other tasks, such as the selection of direction, are also transferred to different functions.

Other functions in the MCACO algorithm allow for the output of data from the MCACO program. The MCACO algorithm produces a file that works with the Tecplot program to create motion movies of the ants searching the domain. Other output from functions include an Excel program file which shows locations of the minimums and a Word program file which provides a detailed report on ant movement.

The most important function in the whole algorithm is the one that actually evaluates the fitness of the functions. Inside this function, the routine which updates the minimum values is contained. The routine activates when a lower minimum fitness for a specific nest is found. Then the old minimum values are overwritten and the new values are stored in memory.

Another key topic related to functions is the amount of function calls. As it currently stands, the first part of the MCACO algorithm consumes a fewer amount of the total number of function calls than the second part. The first sixteen nests use about 43 percent of the total amount of function calls while the final three nests use roughly 57 percent. The following description explains which functions are allocated to which function calls and it provides a few details of the algorithm:

A. First 16 function calls
    a. Evaluates the value of the objective function for each of the nests

B. Function calls from 17 to 1296
    a. Each of the 16 nests runs for 80 iterations
    b. Two degree directions at 45 and 0
    c. Every four iterations degree changes between 45 and 0
    d. Every four iterations values reset, updated, and nest is moved
    e. Every four iterations radius shrinks by 90%
    f. From radius=2 to radius=0.243

C. Function calls from 1297 to 1864
    a. Location with 1st lowest value chosen to start nest
    b. Degree is random between 0 and 90
    c. Every four iterations degree is randomly chosen between 0 to 90
    d. Every four iterations values reset, updated, and nest is moved

e. Every four iterations radius shrinks by 95%

f. From radius=2 to radius=0.001686

D. Function calls from 1865 to 2432

a. Location with 2nd lowest value chosen to start nest

b. Degree is random between 0 and 90

c. Every four iterations degree is randomly chosen between 0 to 90

d. Every four iterations values reset, updated, and nest is moved

e. Every four iterations radius shrinks by 95%

f. From radius=2 to radius=0.001686

E. Function calls from 2433 to 3000

a. Location with 3rd lowest value chosen to start nest

b. Degree is random between 0 and 90

c. Every four iterations degree is randomly chosen between 0 to 90

d. Every four iterations values reset, updated, and nest is moved

e. Every four iterations radius shrinks by 95%

f. From radius=2 to radius=0.001686

The description above explains the intricacies of how the ants move in the single-objective version of the code. However, the multi-objective version of the algorithm is slightly different because it involves the use of the NNC method.

3.7. 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 [25],

$$\begin{aligned} &\text{Problem P1} \\ &\min_{x}\{\mu_1(x) \quad \mu_2(x)\} \\ &\text{subject to:} \\ &g_j(x) \le 0, \quad (1 \le j \le r) \\ &h_k(x) = 0, \quad (1 \le k \le s) \\ &x_{li} \le x_i \le x_{ui}, \quad (1 \le i \le n_x) \end{aligned} \tag{11}$$

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 [25],

$$\begin{aligned} &\text{Problem PU1} \\ &\min_{x}\mu_i(x), \quad (1 \le i \le n) \\ &\text{subject to:} \\ &g_j(x) \le 0, \quad (1 \le j \le r) \\ &h_k(x) = 0, \quad (1 \le k \le s) \\ &x_{li} \le x_i \le x_{ui}, \quad (1 \le i \le n_x) \end{aligned} \tag{12}$$

The line connecting the two anchor points is called the utopia line [25]. The next step is to normalize the search space. Let the utopia point be defined by [25],

$$\mu^u = \left[\mu_1(x^{1*}) \quad \mu_2(x^{2*})\right]^T \tag{13}$$

Also, let the distances between the anchor points and the utopia point be defined by [25],

$$L_1 = \mu_1(x^{2^*}) - \mu_1(x^{1^*}) \text{ and } L_2 = \mu_2(x^{1^*}) - \mu_2(x^{2^*}) \tag{14}$$

The normalized design metrics can now be evaluated as follows [25],

$$\bar{\mu} = \left[\frac{\mu_1(x) - \mu_1(x^{1^*})}{L_1} \qquad \frac{\mu_2(x) - \mu_2(x^{2^*})}{L_2} \right]^T \tag{15}$$

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 [25],

$$\bar{N}_1 = [\bar{\mu}^{2^*} \qquad -\bar{\mu}^{1^*}] \tag{16}$$

The following step is to compute the normalized increments along the utopia line vector which can be calculated as follows [25],

$$\delta_1 = \frac{1}{m_1 - 1} \tag{17}$$

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 [25],

$$\begin{aligned} &\bar{X}_{pj} = \alpha_{1j}\bar{\mu}^{1^*} + \alpha_{2j}\bar{\mu}^{2^*} \\ &\text{where} \\ &0 \le \alpha_{1j} \le 1, \\ &\sum_{k=1}^{2} \alpha_{kj} = 1 \end{aligned} \tag{18}$$

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 [25],

$$
\begin{aligned}
&\text{Problem P2 for } j^{th} \text{ point} \\
&\min_{x} \ \bar{\mu}_2 \\
&\text{subject to:} \\
&g_j(x) \le 0, \quad (1 \le j \le r) \\
&h_k(x) = 0, \quad (1 \le k \le s) \\
&x_{li} \le x_i \le x_{ui} \\
&\bar{N}_1(\bar{\mu} - \bar{X}_{pj})^T \le 0 \\
&\bar{\mu} = \left[\bar{\mu}_1(x) \quad \bar{\mu}_2(x)\right]^T
\end{aligned}
\tag{19}
$$

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 [25],

$$
\mu = \left[\bar{\mu}_1 L_1 + \mu_1(x^{1*}) \qquad \bar{\mu}_2 L_2 + \mu_2(x^{2*})\right]^T \tag{20}
$$

Equation (16) 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. The next section explains some of the statistical techniques used to examine the results.

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 [29],

$$
\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{21}
$$

In equation (18), 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 [29],

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2} \tag{22}$$

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.

CHAPTER 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:

Beale function :

$$f(x,\ y) = \left(1.5 - x\left(1 - y\right)\right)^2 + \left(2.25 - x\left(1 - y^2\right)\right)^2 + \left(2.625 - x\left(1 - y^3\right)\right)^2$$

for $x \in [-4.5,\ 4.5]$ and $y \in [-4.5,\ 4.5]$ (23)

The global minimum is $f(x,\ y) = 0$ located at $(x,\ y) = (3,\ 0.5)$

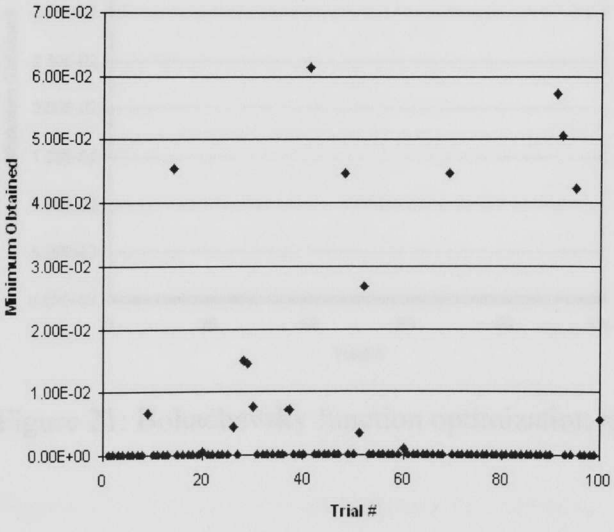


Figure 19: Beale function optimization results

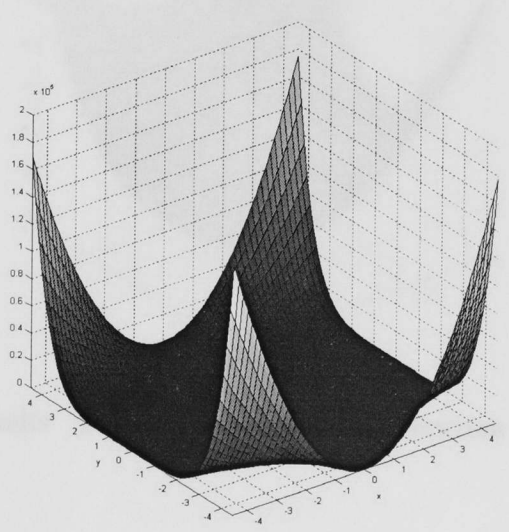


Figure 20: Beale function

Table 7: Beale function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0043947	0.0130699
x location	2.9973173	0.1716337
y location	0.4945017	0.0465925

Bohachevsky function :

$$f(x, y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\cos(4\pi y) + 0.7$$

for $x \in [-10, 10]$ and $y \in [-10, 10]$ (24)

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

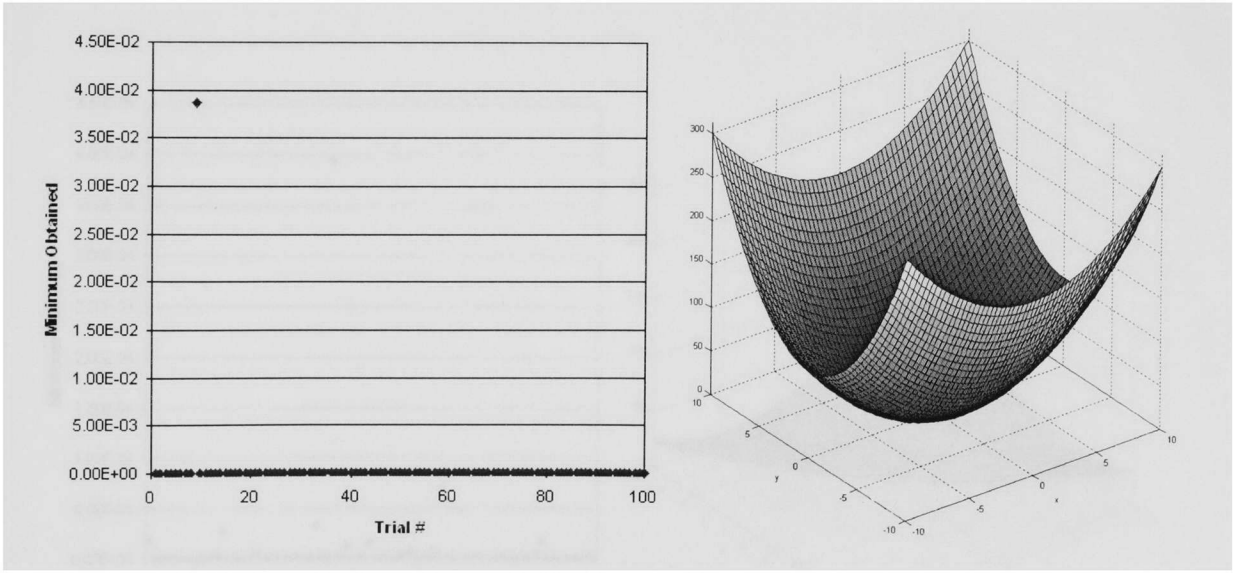


Figure 21: Bohachevsky function optimization results Figure 22: Bohachevsky function

Table 8: Bohachevsky function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0003941	0.0038714
x location	0.0004838	0.0052729
y location	0.0000192	0.0003419

Booth function :

$$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

for $x \in [-10,\ 10]$ and $y \in [-10,\ 10]$ (25)

The global minimum is $f(x,\ y) = 0$ located at $(x,\ y) = (1,\ 3)$

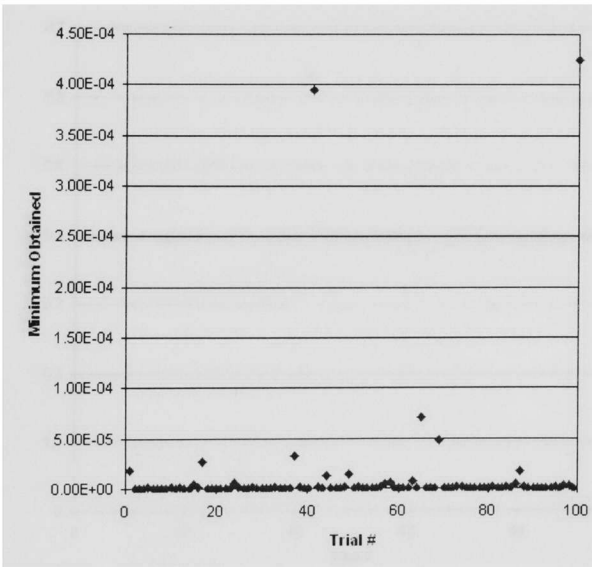


Figure 23: Booth function optimization results

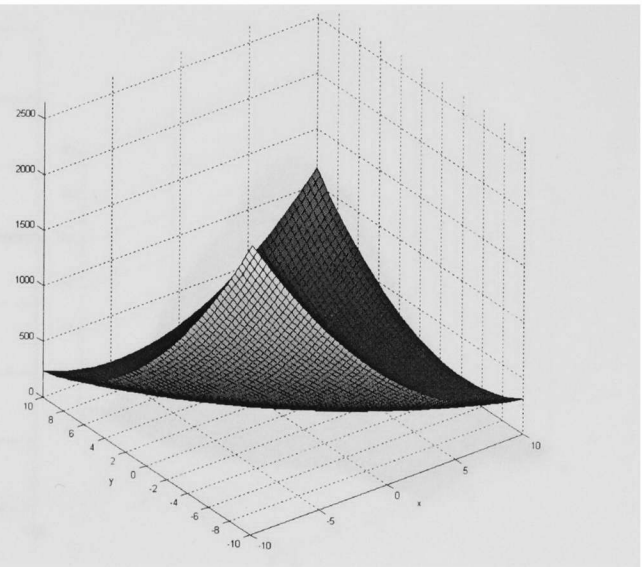


Figure 24: Booth function

Table 9: Booth function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0000113	0.0000580
x location	1.0001123	0.0022938
y location	2.9999238	0.0023718

Branin function :

$$f(x,\ y)=\left(y-\frac{5}{4\pi^2}x^2+5\frac{x}{\pi}-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos(x)+10 \tag{26}$$

for $x \in [-5,\ 10]$ and $y \in [0,\ 15]$

The global minimum is $f(x,\ y)=0.397887$ located at $(x,\ y)=(\pi,\ 2.275)$, $(-\pi,\ 12.275)$, and $(9.42478,\ 2.475)$

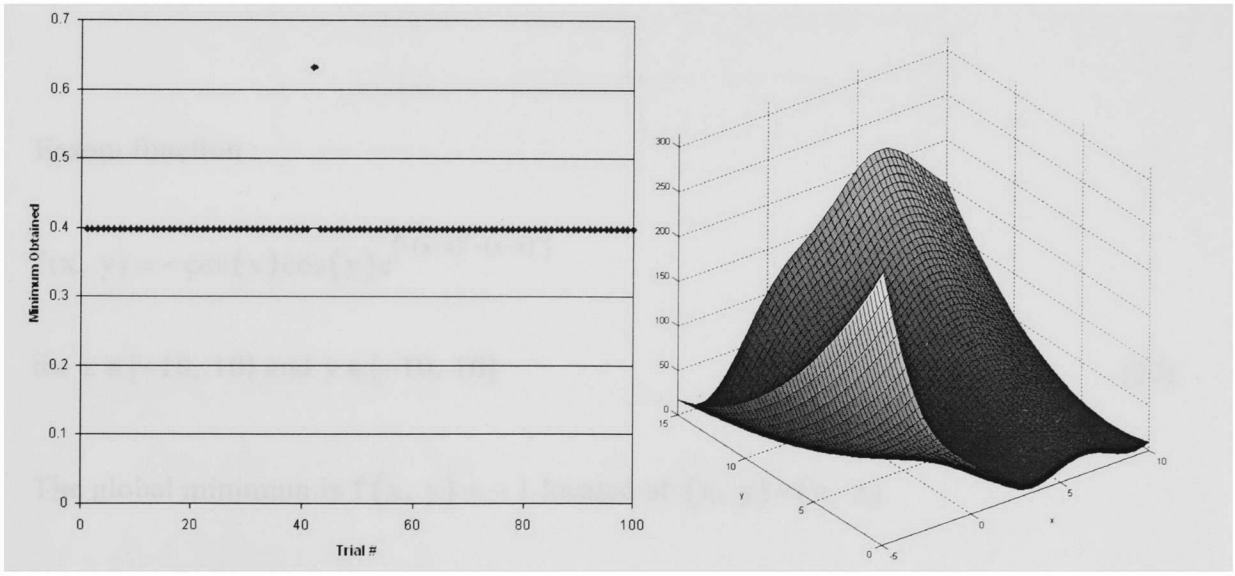


Figure 25: Branin function optimization results

Figure 26: Branin function

Table 10: Branin function optimization with MCACO

	Average	Standard Deviation
Minimum 1 obtained	0.3978878	0.0000006
x1 location	-3.1415813	0.0002687
y1 location	12.2501533	0.0009425
Minimum 2 obtained	0.4065537	0.0450241
x2 location	3.1445119	0.0157033
y2 location	2.2311015	0.0988626
Minimum 3 obtained	0.3978879	0.0000003
x3 location	9.4247669	0.0002363
y3 location	2.2499074	0.0004936

Easom function :

$$f(x, y) = -\cos(x)\cos(y)e^{\left(-(x-\pi)^2-(y-\pi)^2\right)}$$

for $x \in [-10, 10]$ and $y \in [-10, 10]$ (27)

The global minimum is $f(x, y) = -1$ located at $(x, y) = (\pi, \pi)$

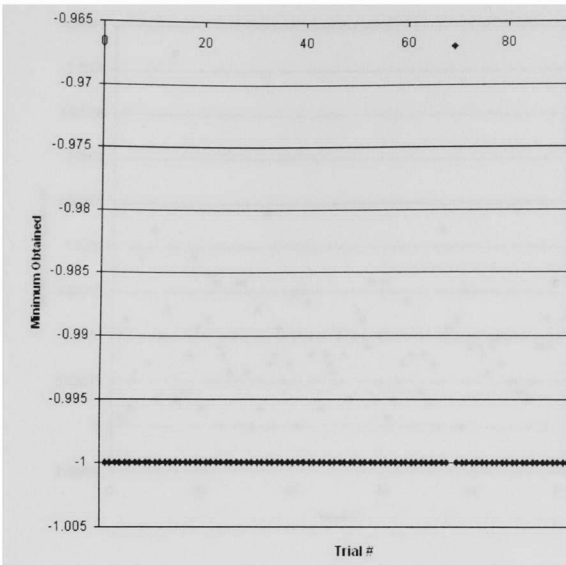


Figure 27: Easom function optimization results

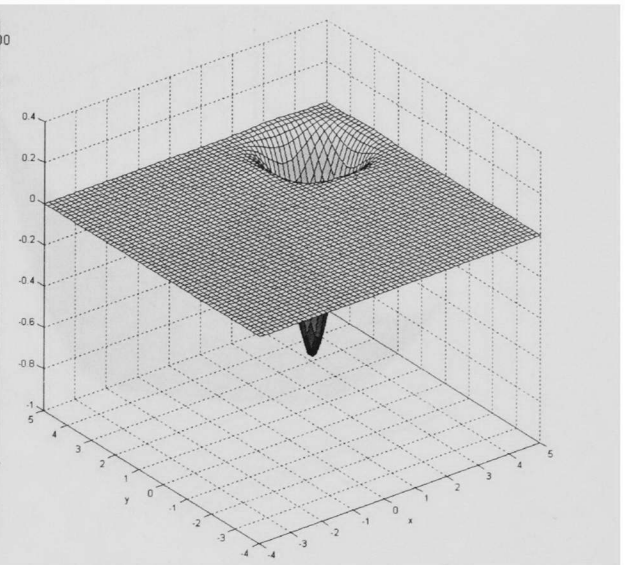


Figure 28: Easom function

Table 11: Easom function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	-0.9996689	0.0033062
x location	3.1416097	0.0004682
y location	3.1401308	0.0149689

Goldstein and Price (GP) function

$$f(x,y)=\left(1+\left(x+y+1\right)^2\left(19-14x+3x^2-14y+6xy+3y^2\right)\right)$$
$$\left(30+\left(2x-3y\right)^2\left(18-32x+12x^2+48y-36xy+27y^2\right)\right)$$

for $x \in [-2,\ 2]$ and $y \in [-2,\ 2]$ (28)

The global minimum is $f(x,\ y)=3$ located at $(x,\ y)=(0,\ -1)$

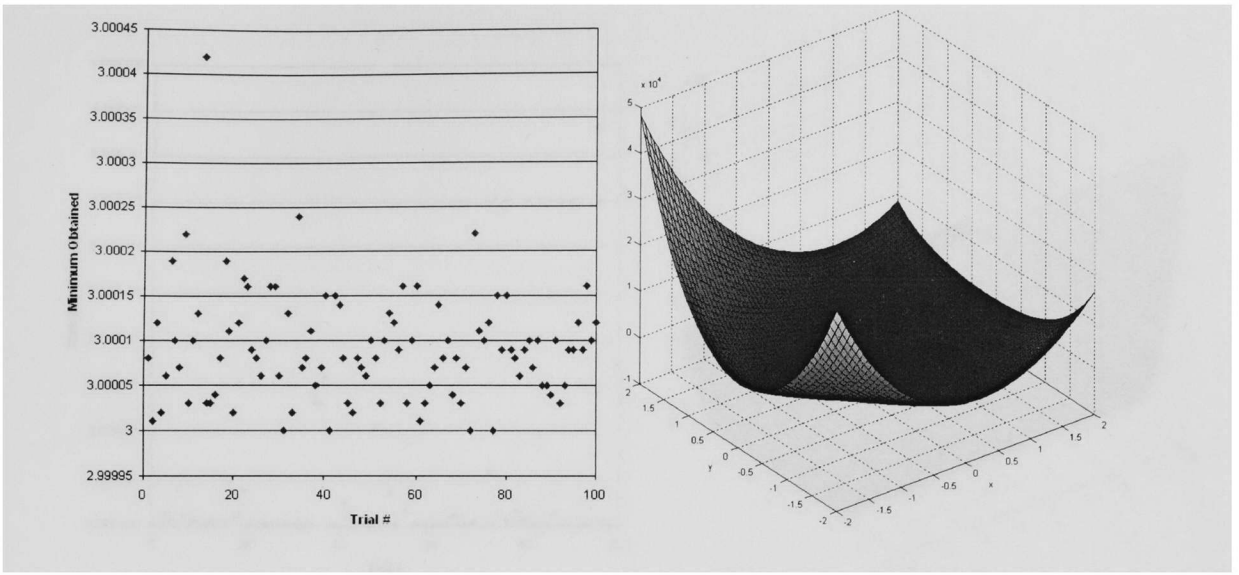


Figure 29: GP function optimization results

Figure 30: GP function

Table 12: Goldstein and Price function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	3.0000918	0.0000614
x location	0.0000179	0.0004789
y location	-1.0000065	0.0003307

Freudenstein and Roth (FR) function :

$$f(x,\ y) = \left(-13 + x + \left(\left(5 - y\right)y - 2\right)y\right)^2 + \left(-29 + x + \left(\left(y + 1\right)y - 14\right)y\right)^2$$

for $x \in [-8,\ 8]$ and $y \in [-8,\ 8]$ (29)

The global minimum is $f\left(x,\ y\right) = 0$ located at $\left(x,\ y\right) = \left(5,\ 4\right)$

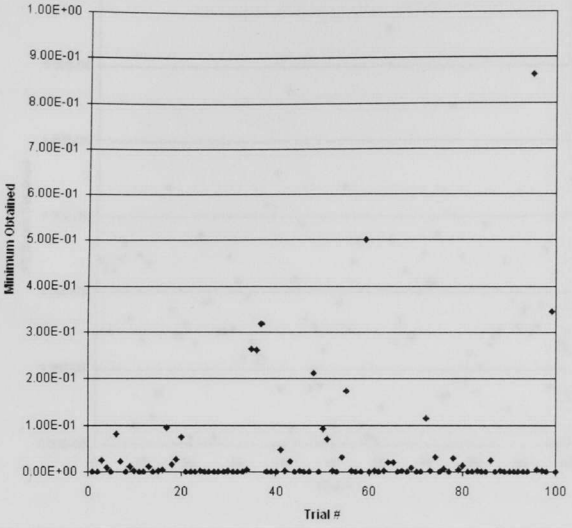


Figure 31: FR function optimization results

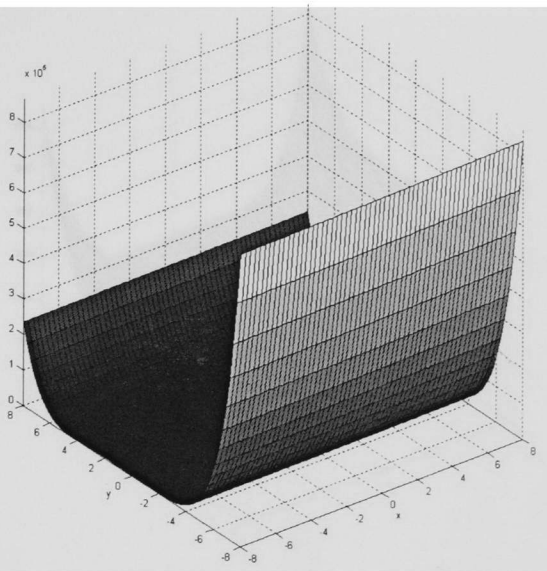


Figure 32: FR function

Table 13: Freudenstein and Roth function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0390358	0.1161078
x location	4.9919502	0.1643608
y location	4.0001385	0.0029115

Hump function :

$$f(x,\ y) = 1.0316285 + 4x^2 - 2.1x^4 + \frac{x^6}{3} + xy - 4y^2 + 4y^4$$

for $x \in [-5,\ 5]$ and $y \in [-5,\ 5]$ (30)

The global minimum is $f(x,\ y) = 0$ located at $(x,\ y) = (0.0898,\ -0.7126)$ and $(-0.0898,\ 0.7126)$

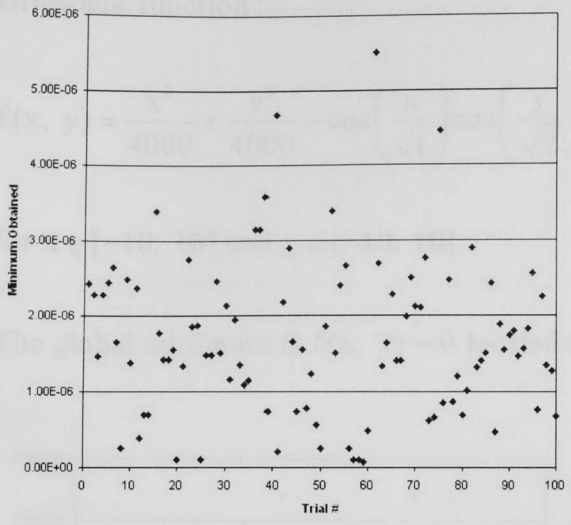


Figure 33: Hump function optimization results

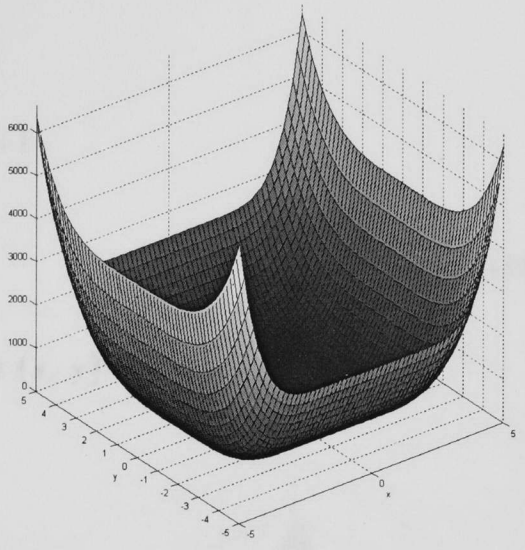


Figure 34: Hump function

Table 14: Hump function optimization with MCACO

	Average	Standard Deviation
Minimum 1 obtained	0.0000017	0.0000010
x1 location	0.0897343	0.0005088
y1 location	-0.7126670	0.0002812
Minimum 2 obtained	0.0000017	0.0000011
x2 location	-0.0899025	0.0005251
y2 location	0.7126682	0.0002704

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 $y \in [-10, 10]$ (31)

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

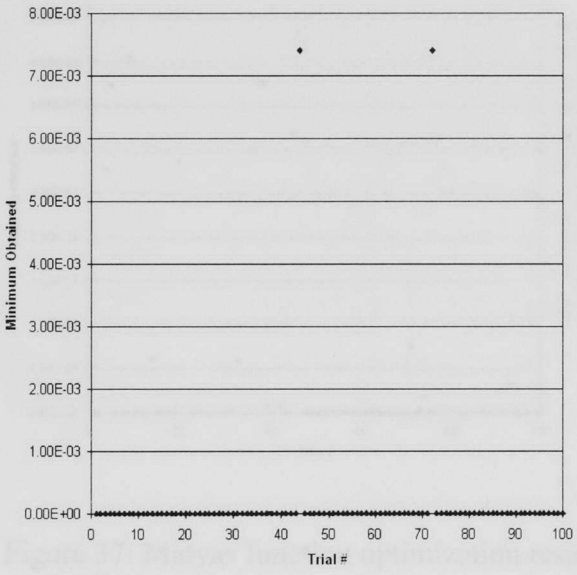


Figure 35: Griewank function optimization results

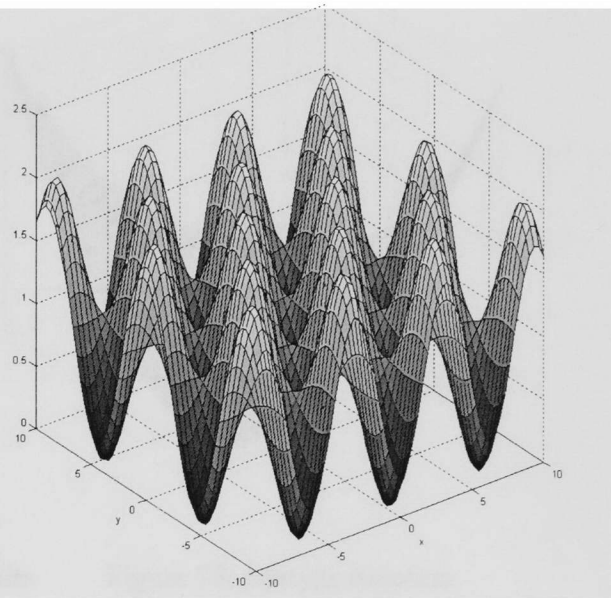


Figure 36: Griewank function

Table 15: Griewank function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0001481	0.0010407
x location	-0.0627627	0.4418653
y location	0.0001186	0.6308250

Matyas function :

$$f(x, y) = 0.26\left(x^2 + y^2\right) - 0.48xy$$

for $x \in [-10, 10]$ and $y \in [-10, 10]$ (32)

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

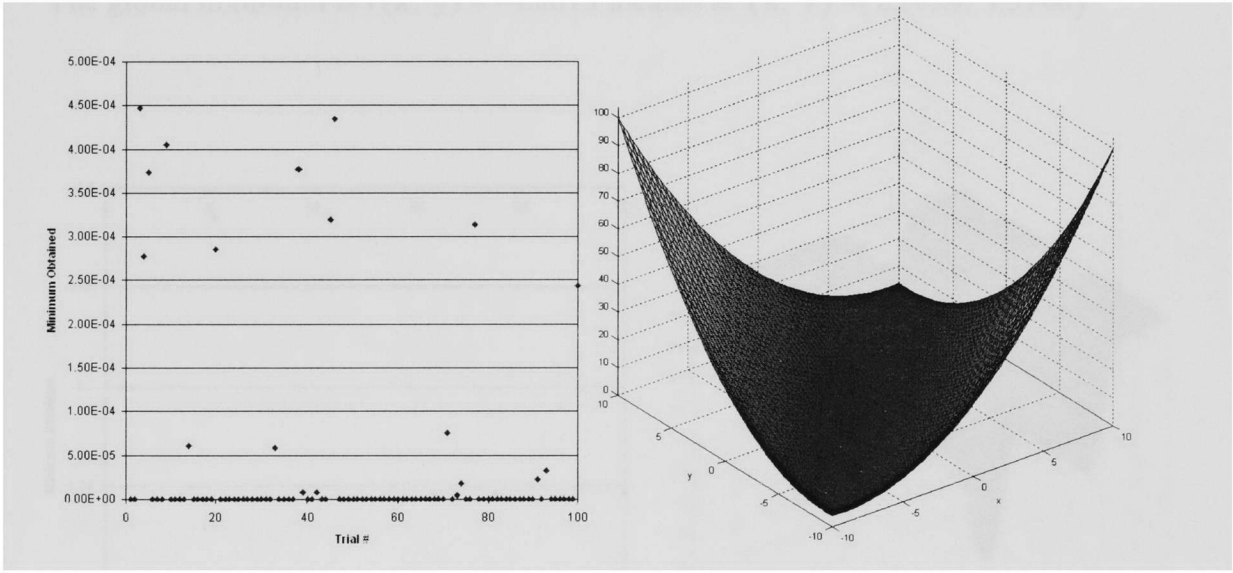


Figure 37: Matyas function optimization results

Figure 38: Matyas function

Table 16: Matyas function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0000377	0.0001067
x location	0.0011115	0.0306754
y location	0.0011233	0.0309074

Michalewics function :

$$f(x,\ y) = -\sum_{i=1}^{2} \sin(x_i)\left(\sin\left(\frac{ix_i^{\ 2}}{\pi}\right)\right)^{20}$$

for $x \in [0,\ \pi]$ and $y \in [0,\ \pi]$ (33)

The global minimum is $f(x,\ y) = -1.8013$ located at $(x,\ y) = (2.2029,\ 1.5708)$

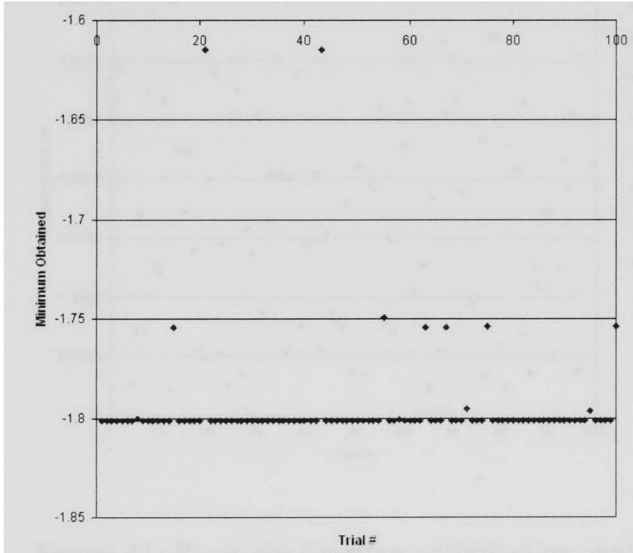


Figure 39: Michalewics function optimization results

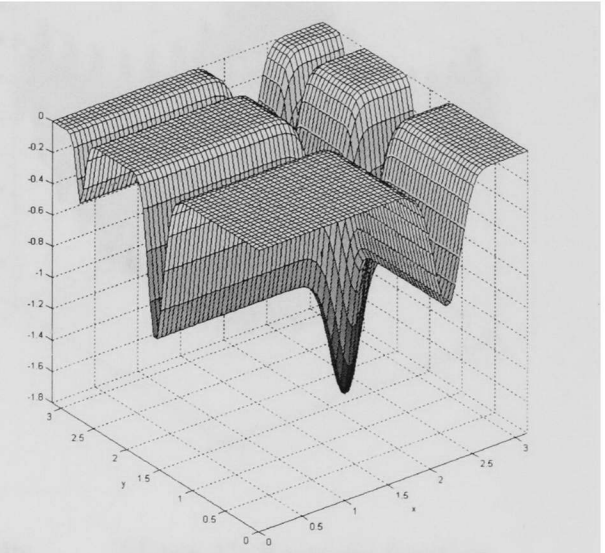


Figure 40: Michalewics function

Table 17: Michalewics function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	-1.7945548	0.0282537
x location	2.2036469	0.0046191
y location	2.8580689	2.5890481

Rastrigin function :

$$f(x,\ y) = 20 + \sum_{i=1}^{2}\left(x_i^{\ 2} - 10\cos\left(2\pi x_i\right)\right)$$

for $x \in [-5.12,\ 5.12]$ and $y \in [-5.12,\ 5.12]$ (34)

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

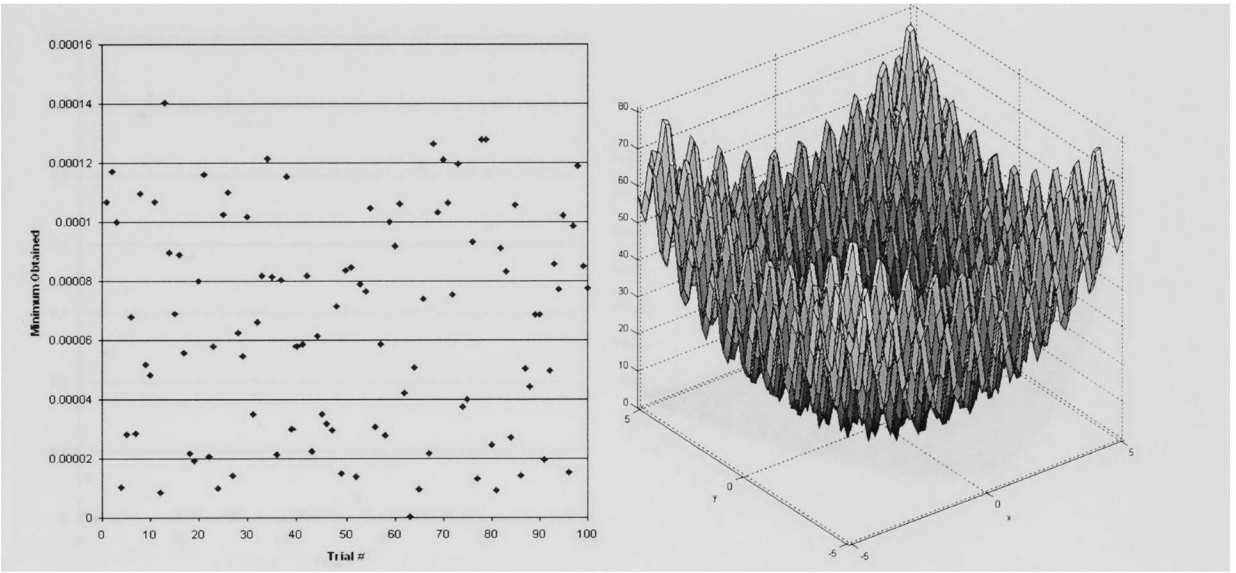


Figure 41: Rastrigin function optimization results

Figure 42: Rastrigin function

Table 18: Rastrigin function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0000656	0.0000368
x location	0.0000148	0.0003758
y location	-0.0000660	0.0004338

Rosenbrock function :

$$f(x,\ y) = 100\left(y - x^2\right)^2 + \left(1 - x\right)^2$$

for $x \in [-2.048,\ 2.048]$ and $y \in [-2.048,\ 2.048]$ (35)

The global minimum is $f(x,\ y) = 0$ located at $(x,\ y) = (1,\ 1)$

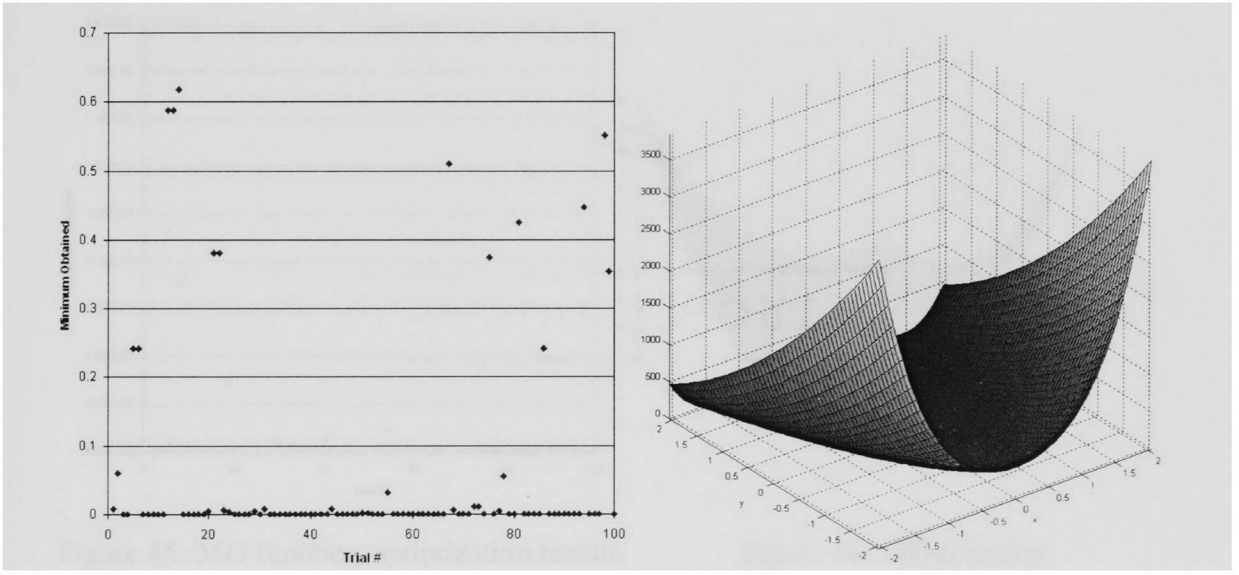


Figure 43: Rosenbrock function optimization results

Figure 44: Rosenbrock function

Table 19: Rosenbrock function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0617681	0.1548584
x location	0.9327005	0.2400674
y location	0.9265443	0.3730122

Martin and Gaddy (MG) function :

$$f(x, y) = (x - y)^2 + \left(\frac{x + y - 10}{3}\right)^2$$

for $x \in [0, 10]$ and $y \in [0, 10]$ (36)

The global minimum is $f(x, y) = 0$ located at $(x, y) = (5, 5)$

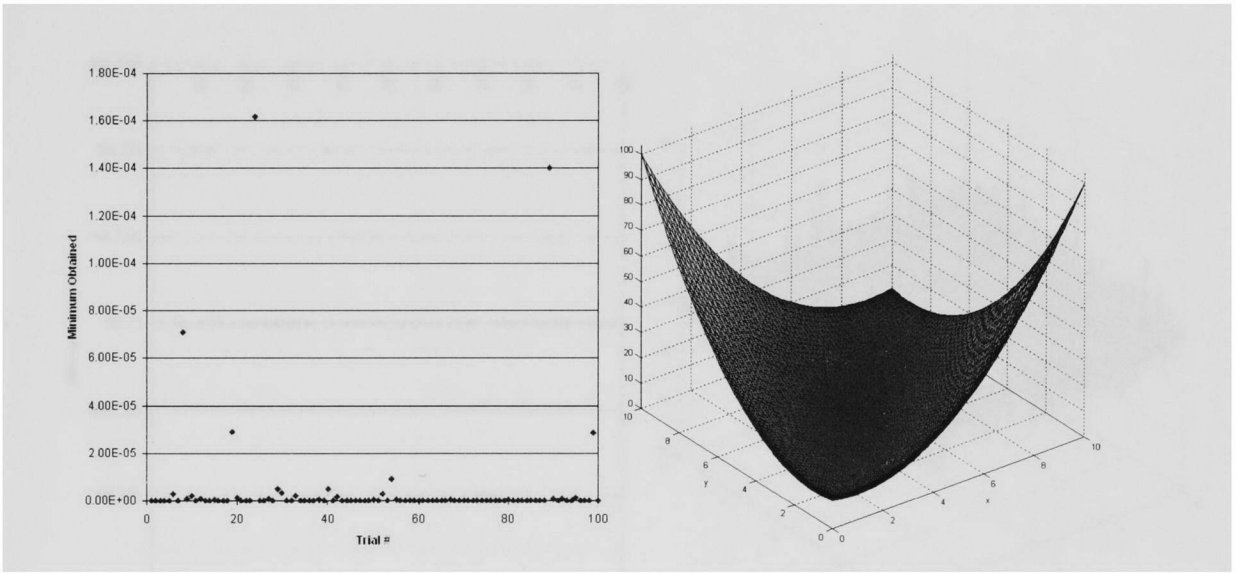


Figure 45: MG function optimization results

Figure 46: MG function

Table 20: Martin and Gaddy function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0000048	0.0000225
x location	4.9997063	0.0033152
Y location	4.9997602	0.0030853

Shubert function :

$$f(x, y) = \left(\sum_{i=1}^{5} i\cos\left((i+1)x + i\right) \right) \left(\sum_{i=1}^{5} i\cos\left((i+1)y + i\right) \right)$$

for $x \in [-10, 10]$ and $y \in [-10, 10]$ (37)

The global minimum is $f(x, y) = -186.7309$ located at eighteen different locations

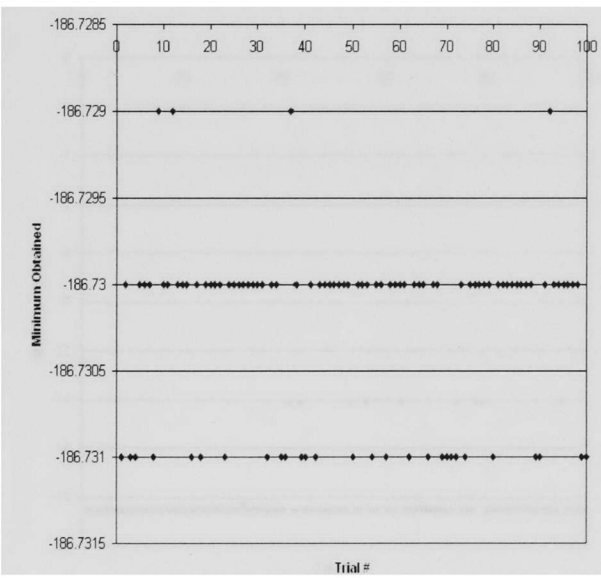


Figure 47: Shubert function optimization results

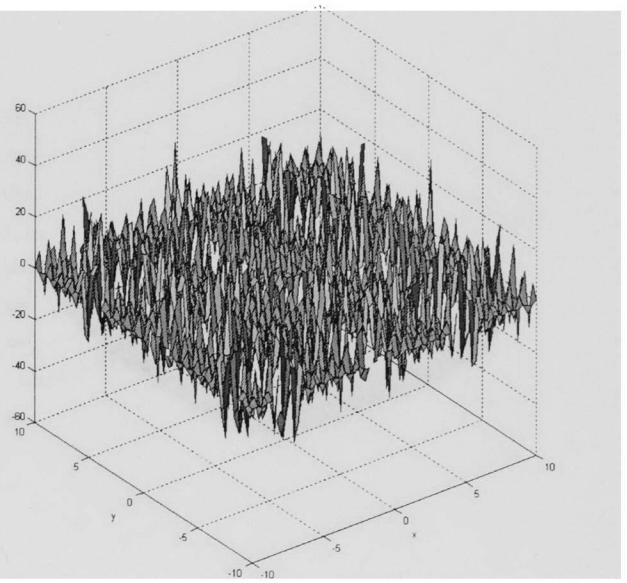


Figure 48: Shubert function

Table 21: Shubert function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	-186.7302400	0.0005150
x location	1.6702681	5.6381038
y location	-2.0108194	6.8626155

Rosen function :

$$f(x, y) = 0.25x^4 - 3x^3 + 11x^2 - 13x + 0.25y^4 - 3y^3 + 11y^2 - 13y$$

for $x \in [-10, 10]$ and $y \in [-10, 10]$ (38)

The global minimum is $f(x, y) = -18.5680$ located at $(x, y) = (5.3301, 5.3301)$

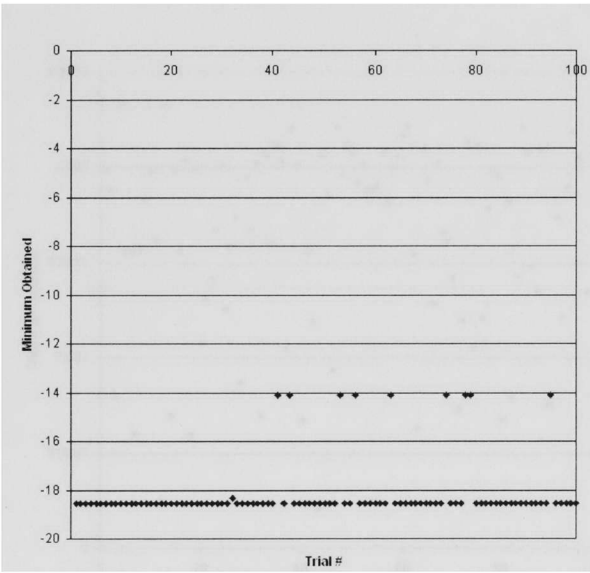


Figure 49: Rosen function optimization results

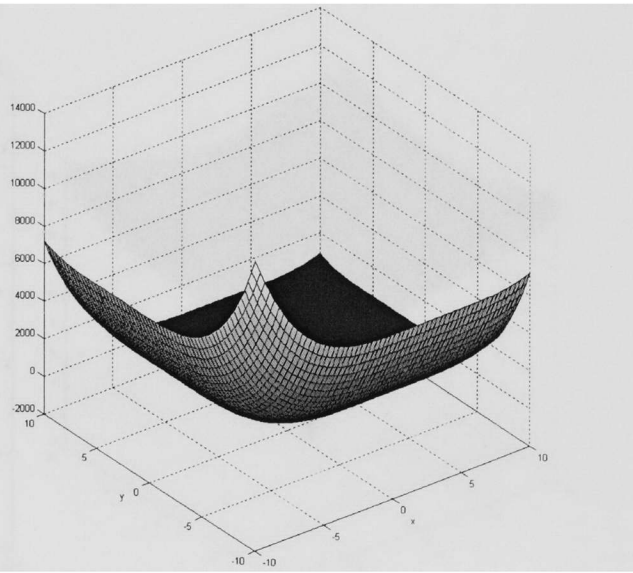


Figure 50: Rosen function

Table 22: Rosen function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	-18.1634670	1.2844459
x location	5.2386992	0.6273756
y location	5.0180015	1.1433247

Ackley function :

$$f(x,\ y) = -20e^{\left(-0.2\sqrt{\frac{1}{2}\sum_{i=1}^{2} x_i^{\,2}}\right)} - e^{\left(\frac{1}{2}\sum_{i=1}^{2}\cos(2\pi x_i)\right)} + 20 + e^{1}$$

for $x \in [-10,\ 10]$ and $y \in [-10,\ 10]$ (39)

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

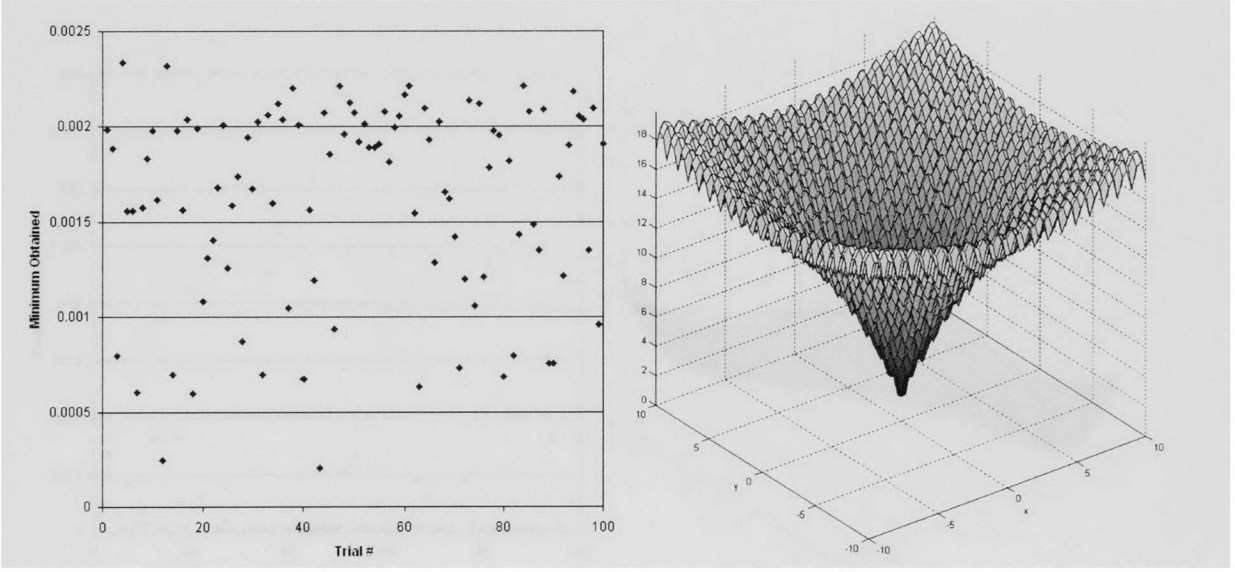


Figure 51: Ackley function optimization results

Figure 52: Ackley function

Table 23: Ackley function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0016163	0.0005241
x location	0.0000166	0.0004229
y location	0.0000086	0.0004248

Perm #1 function :

$$f(x,\ y)=\sum_{k=1}^{2}\left[\sum_{i=1}^{2}\left(i^{k}+50\right)\left(\left(\frac{x_i}{i}\right)^{k}-1\right)\right]^2$$

for $x \in [-2,\ 2]$ and $y \in [-2,\ 2]$ (40)

The global minimum is $f(x,\ y)=0$ located at $(x,\ y)=(1,\ 2)$

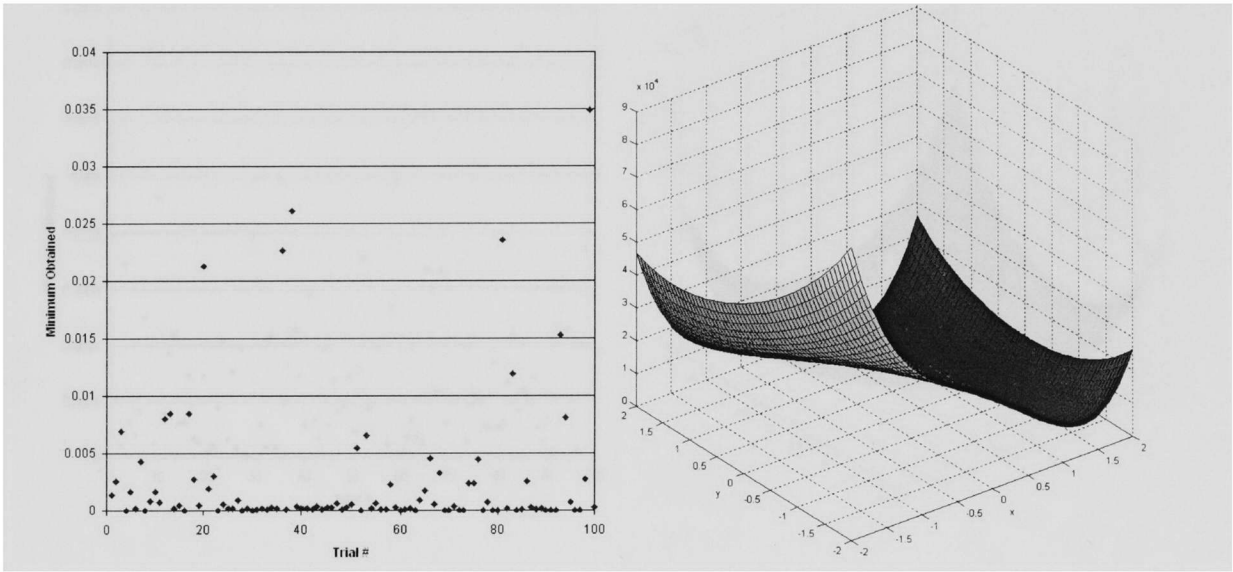


Figure 53: Perm #1 function optimization results

Figure 54: Perm #1 function

Table 24: Perm #1 function optimization with MCACO

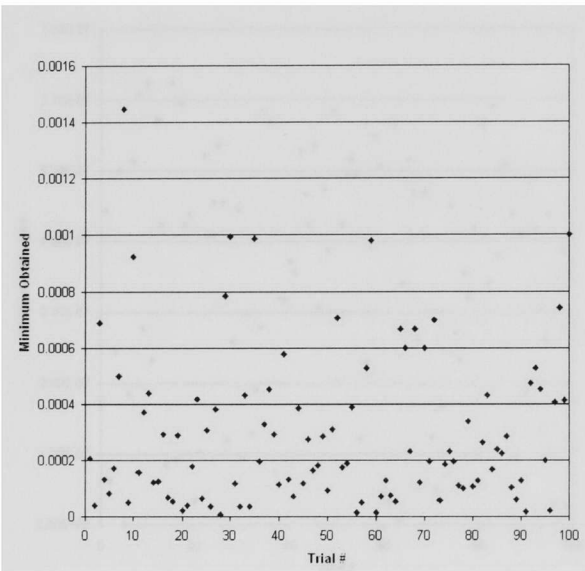
	Average	Standard Deviation
Minimum obtained	0.0026820	0.0060401
x location	1.0227850	0.0298473
y location	1.9544575	0.0587214

Perm #2 function :

$$f(x,\ y) = \sum_{k=1}^{2}\left[\sum_{i=1}^{2}(i+50)\left(x_i^{\ k} - i^{-k}\right)\right]^2$$

for $x \in [-2,\ 2]$ and $y \in [-2,\ 2]$ (41)

The global minimum is $f(x,\ y) = 0$ located at $(x,\ y) = (1,\ 0.5)$



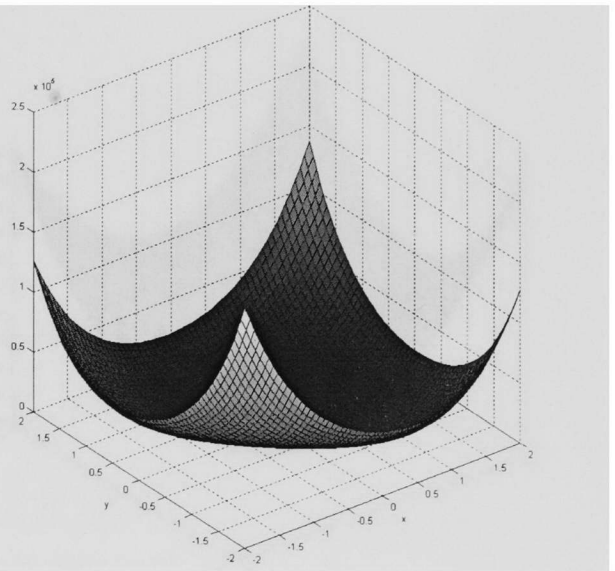


Figure 55: Perm #2 function optimization results

Figure 56: Perm #2 function

Table 25: Perm #2 function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0002972	0.0002778
x location	0.7324195	0.2532845
y location	0.7624149	0.2484109

Sphere function :

$$f(x,\ y) = \sum_{i=1}^{2} x_i^{\,2}$$

for $x \in [-5.12,\ 5.12]$ and $y \in [-5.12,\ 5.12]$ (42)

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

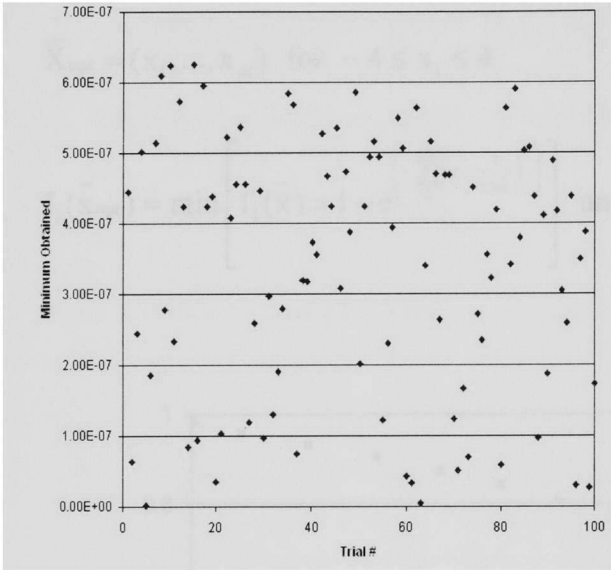


Figure 57: Sphere function optimization results

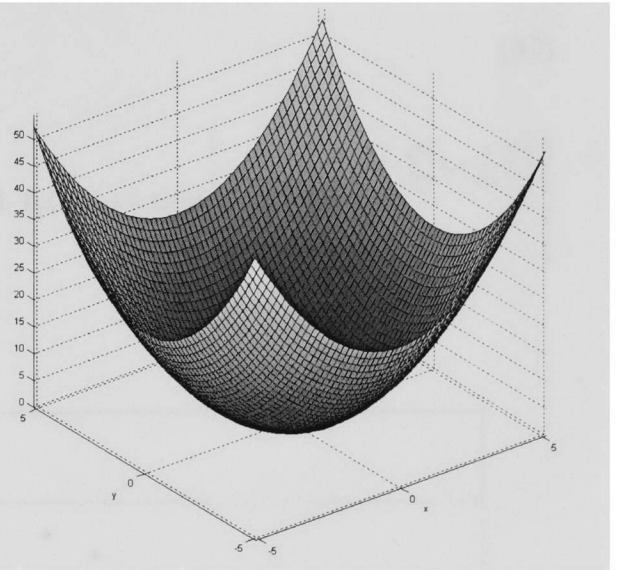


Figure 58: Sphere function

Table 26: Sphere function optimization with MCACO

	Average	Standard Deviation
Minimum obtained	0.0000003	0.0000002
x location	-0.0000151	0.0004330
y location	-0.0000340	0.0003944

4.2. Multi-Objective Optimization Test Cases

The results for the multi-objective test cases have been obtained using the MCACO algorithm in conjunction with the NNC method. For each function, the routine was run 50 different times and the results for the best case were recorded. The functions and results obtained are given below:

Fonseca and Fleming two – objective test problem :

$$\vec{X}_{opt} = (x_1, \ldots, x_m) \quad \text{for} \quad -4 \le x_i \le 4 \tag{43}$$

$$f_1(\vec{x}_{opt}) = \min\left[f_1(\vec{x}) = 1 - e^{\left(-\sum_{i=1}^{m} \left[x_i - \frac{1}{\sqrt{m}} \right]^2 \right)} \right] \quad \text{and} \quad f_2(\vec{x}_{opt}) = \min\left[f_2(\vec{x}) = 1 - e^{\left(-\sum_{i=1}^{m} \left[x_i + \frac{1}{\sqrt{m}} \right]^2 \right)} \right]$$

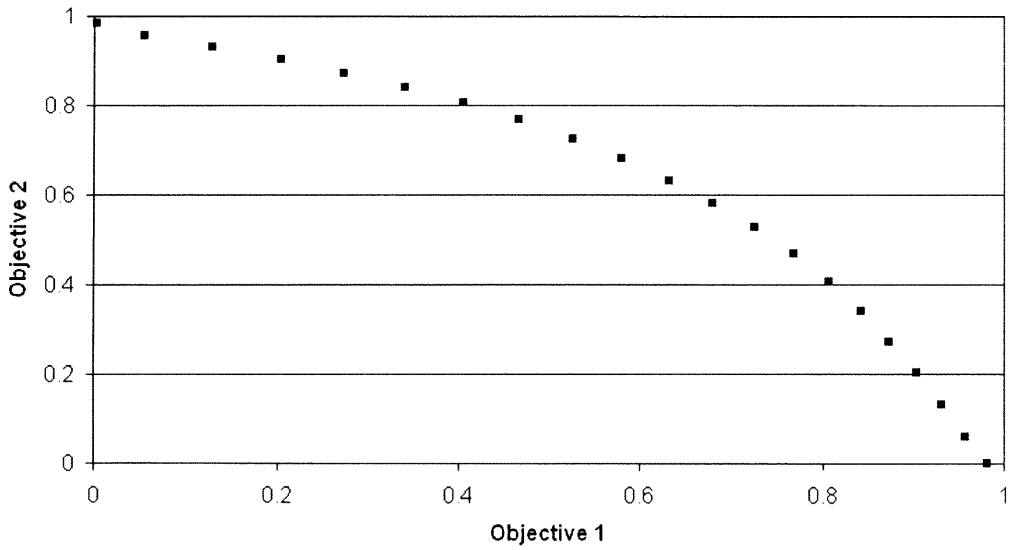


Figure 59: Fonseca and Fleming function optimization results with MCACO

Poloni two – objective test problem :

$$\vec{X}_{opt} = (x_1, ..., x_m) \text{ for } -3.1416 \le x_i \le 3.1416$$

$$f_1(\vec{x}_{opt}) = \max\left[f_1(\vec{x}) = -\left(1 + (A_1 - B_1)^2 + (A_2 - B_2)^2\right)\right]$$

$$f_2(\vec{x}_{opt}) = \max\left[f_2(\vec{x}) = -\left((x+3)^2 + (y+1)^2\right)\right]$$

$$A_1 = 0.5\sin(1) - 2\cos(1) + \sin(2) - 1.5\cos(2) \quad (44)$$

$$A_2 = 1.5\sin(1) - \cos(1) + 2\sin(2) - 0.5\cos(2)$$

$$B_1 = 0.5\sin(x) - 2\cos(x) + \sin(y) - 1.5\cos(y)$$

$$B_2 = 1.5\sin(x) - \cos(x) + 2\sin(y) - 0.5\cos(y)$$

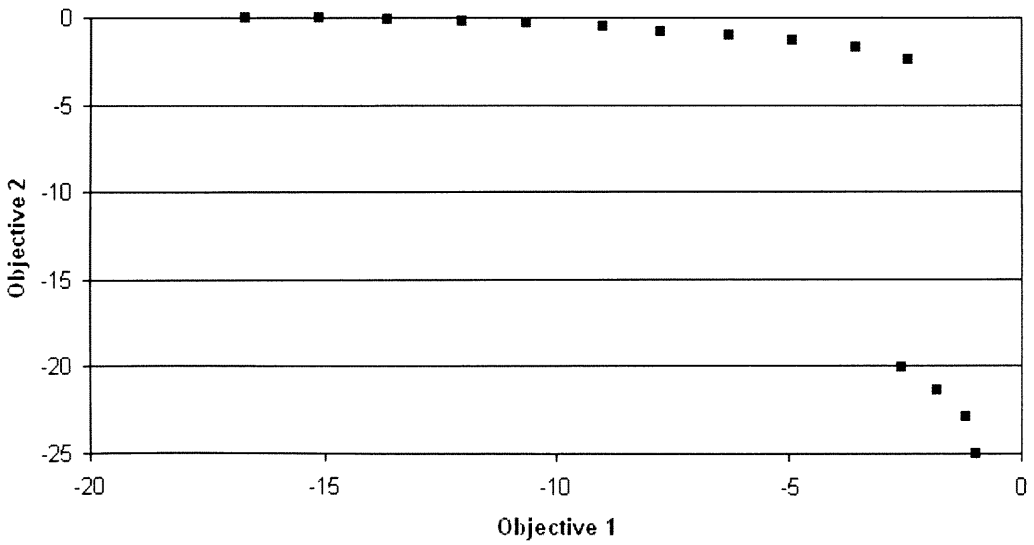


Figure 60: Poloni function optimization results with MCACO

Binh two – objective test problem :

$$\vec{X}_{opt} = (x_1, ..., x_m) \quad \text{for} \quad -5 \le x_i \le 10$$

$$f_1(\vec{x}_{opt}) = \min\left[f_1(\vec{x}) = x^2 + y^2 \right] \quad \text{and} \quad f_2(\vec{x}_{opt}) = \min\left[f_2(\vec{x}) = (x-5)^2 + (y-5)^2 \right] \tag{45}$$

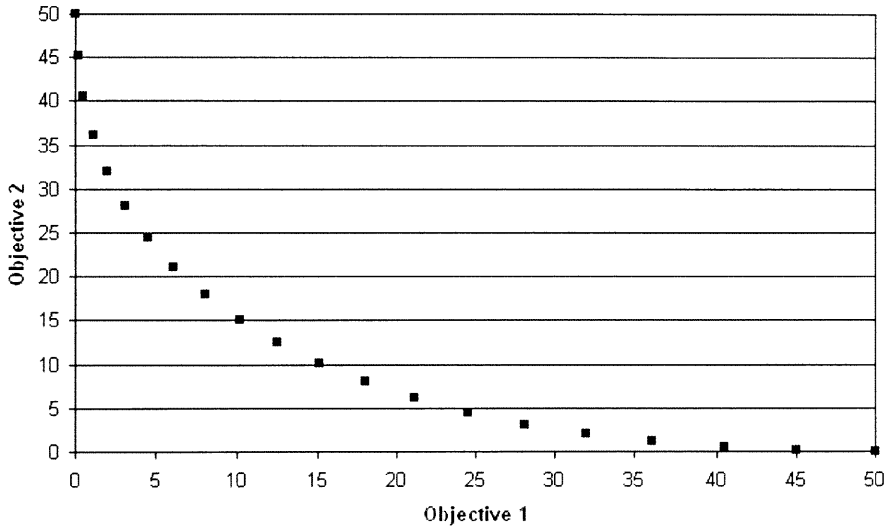


Figure 61: Binh function optimization results with MCACO

Lis two – objective test problem :

$$\vec{X}_{opt} = (x_1, ..., x_m) \quad \text{for} \quad -5 \le x_i \le 10 \tag{46}$$

$$f_1(\vec{x}_{opt}) = \min\left[f_1(\vec{x}) = \sqrt[8]{x^2 + y^2} \right] \quad \text{and} \quad f_2(\vec{x}_{opt}) = \min\left[f_2(\vec{x}) = \sqrt[4]{(x-0.5)^2 + (y-0.5)^2} \right]$$

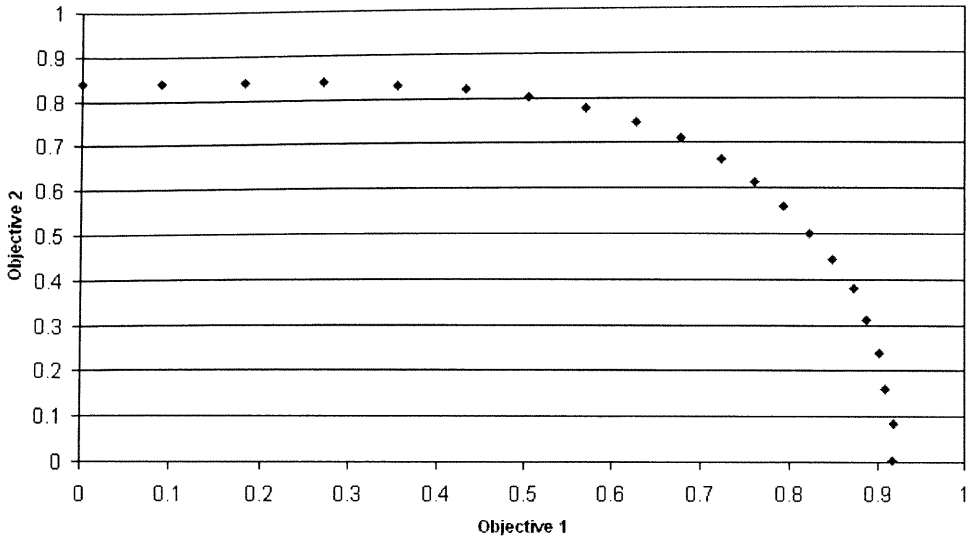


Figure 62: Lis function optimization results with MCACO

Rendon two – objective test problem :

$$\vec{X}_{opt} = (x_1, ..., x_m) \quad \text{for} \quad -3 \le x_i \le 3 \tag{47}$$

$$f_1(\vec{x}_{opt}) = \min\left[f_1(\vec{x}) = \frac{1}{x^2 + y^2 + 1} \right] \quad \text{and} \quad f_2(\vec{x}_{opt}) = \min\left[f_2(\vec{x}) = x^2 + 3y^2 + 1 \right]$$

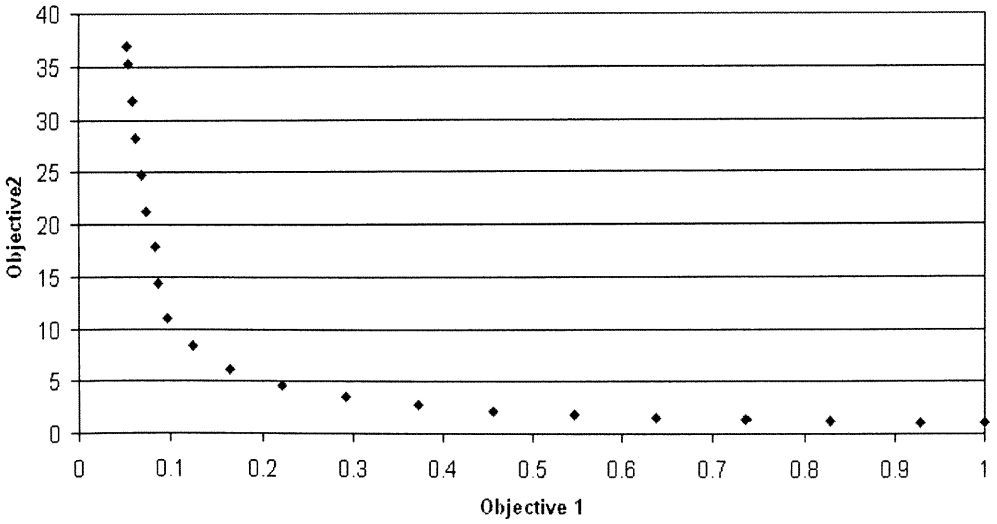


Figure 63: Rendon function optimization results with MCACO

CHAPTER 5

DISCUSSION

5.1. Results and Comparison

The goal of this research was to create and modify an ant colony algorithm that works for continuous functions. The created algorithm applies to both single-objective and multi-objective problems. Many optimization problems, such as the Griewank function, have extreme topologies that the developed algorithm is able to handle well.

Overall, the results for the single-objective cases are excellent. 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. For example, examining table 6, it can be noted that the standard deviation of the minimum obtained, of the x value, and of the y value is small. This means the optimized values are very stable. 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. In table 19, the standard deviation for the y values is relatively large when compared to the magnitude of the average value. This means that some of the y values wobble slightly around the true location of the minimum. The following table shows the real minimum of the functions as compared to the averaged minimum obtained using the MCACO algorithm:

Table 27: Comparison of minimums obtained using MCACO

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 averaged MCACO minimum values for each function closely resemble the actual global minimums. This proves the MCACO algorithm to be accurate.

The number of function calls for all single-objective functions is three thousand function calls. MCACO is a different sort of optimization algorithm and varies from many others because it works by placing pheromone on the actual function topology. The number of function calls being constant has both advantages and drawbacks. The advantage is that even for complicated functions such as the Griewank function, the number of function calls is still three thousand. The disadvantage is that for easier functions, such as the sphere function, the number of function calls is still three thousand and no less. The following table shows how the MCACO algorithm compares to other ant related optimization schemes in terms of function calls:

Table 28: Ant colony based algorithm comparison

Function Name	MCACO	CACO	CIAC	Binary Ant System	DACO	ACO_R
Goldstein and Price	3000	5330	23391	2317.54	229.53	384
Rosenbrock	3000	6842	11797	2580.53	1946.77	820
Reference	-	[31]	[31]	[31]	[32]	[14]

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. For reference and to see how different the ant based algorithms are, a brief overview of the ACO_R method is given.

The best performing ant based algorithm 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]. ACO_R was co-created by the original designer of the ACO, Marco Dorigo. 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). A PDF may be any function such that [14],

$$\int_{-\infty}^{\infty} P(x)dx = 1 \tag{48}$$

The most common PDF is the Gaussian function. As a result, ACO_R uses a Gaussian kernel, which is a weighted sum of several one-dimensional Gaussian functions. This kernel is denoted as follows [14],

$$G^{i}(x) = \sum_{l=1}^{k} \omega_l g_l^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} e^{-\frac{(x-\mu_l^i)^2}{2\sigma_l^{i2}}} \tag{49}$$

In ACO_R, the ants sample a PDF, such as equation (49), 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.

The next table shows the number of function calls for a few optimization algorithms that are not based on ant colony metahueristics [33]:

Table 29: Non-ant based algorithm comparison

Function Name	Continuous Genetic Algorithm	Enhanced Continuous Tabu Search	Enhanced Simulated Annealing
Goldstein and Price	410	231	783
Rosenbrock	960	480	796

The data shows that MCACO does not compare well to other optimization routines. 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. The unique features of the MCACO algorithm make it suitable for certain functions.

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. The results are given in the figures that follow:

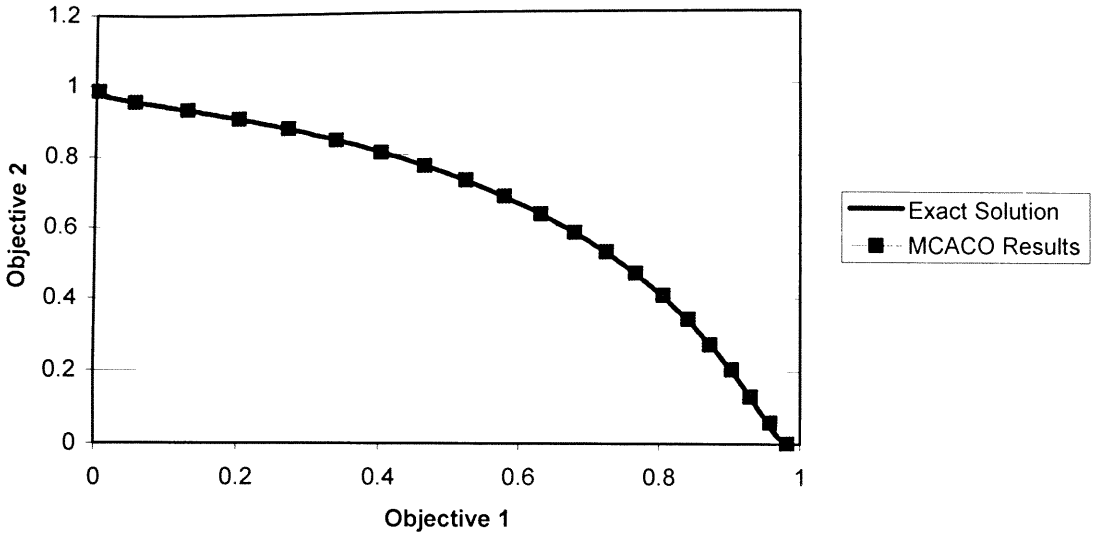


Figure 64: MCACO Fonseca and Flemming comparison with exact solution

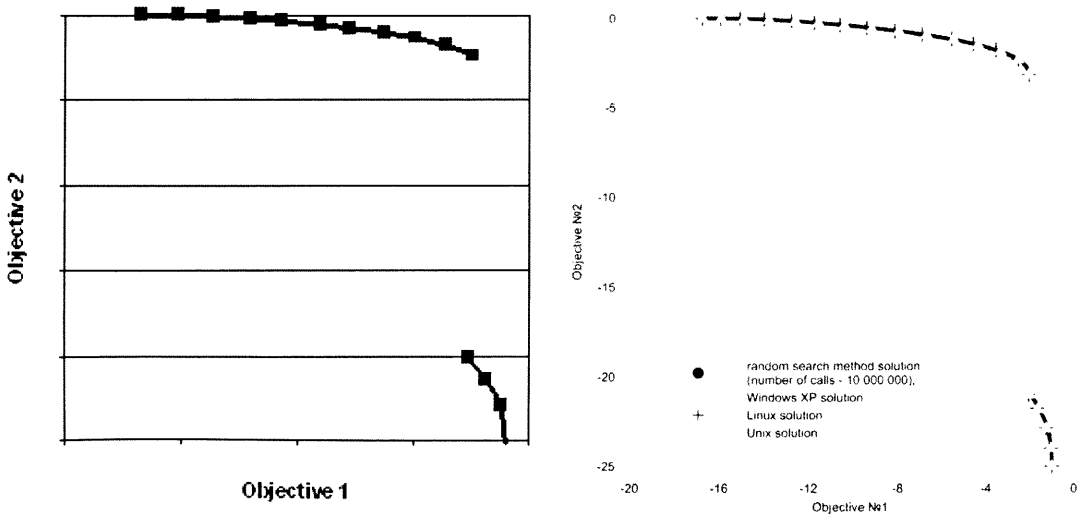


Figure 65: MCACO Poloni comparison with IOSO solution

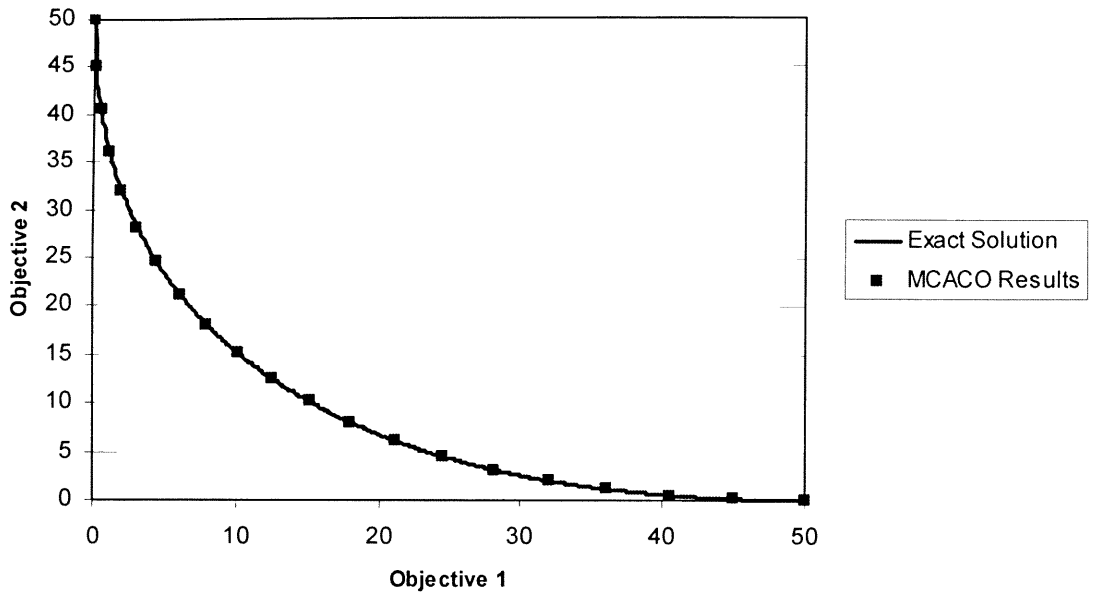


Figure 66: MCACO Binh comparison with exact solution

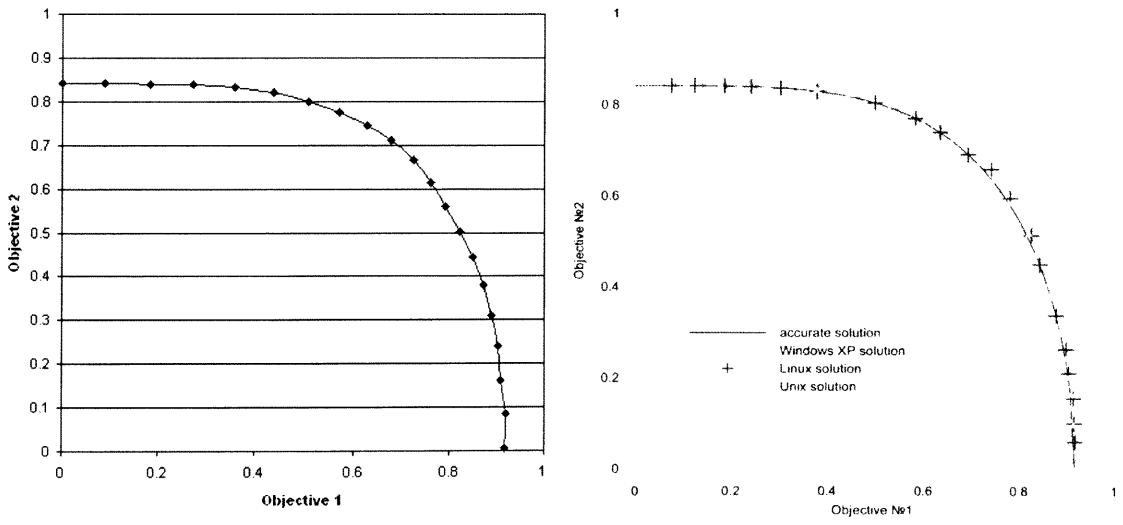


Figure 67: MCACO Lis comparison with IOSO solution

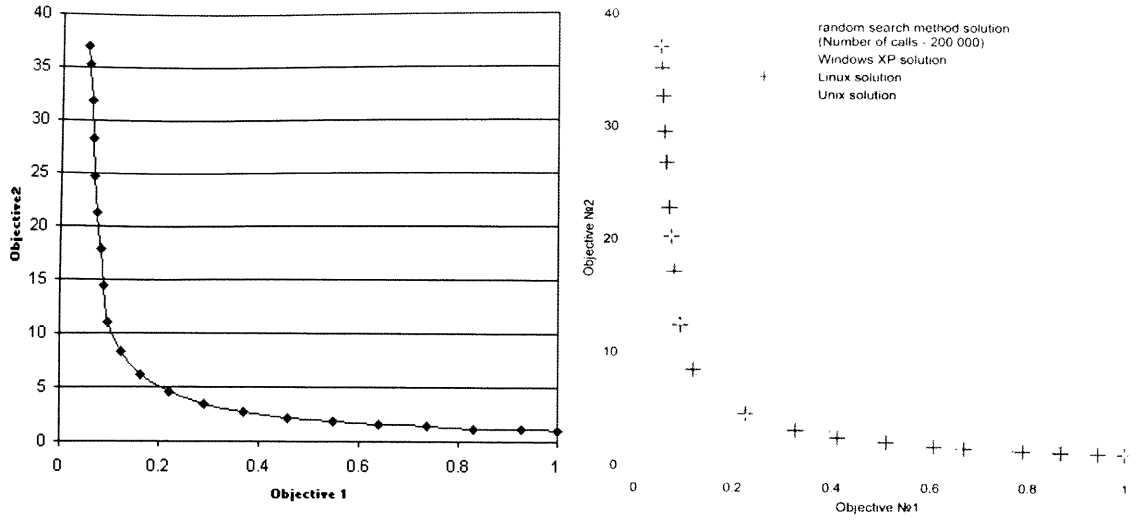


Figure 68: MCACO Rendon comparison with IOSO solution

For the Fonseca and Flemming and the Binh multi-objective functions, the obtained Pareto points lie perfectly on the exact solution. The MCACO results for the Poloni, Lis, and Rendon multi-objective functions are compared to results obtained by using the IOSO NM optimization tool developed by Egorov [30]. The results using the MCACO algorithm are depicted in the figures to the left and the solutions obtained using the IOSO optimization tool are given in the figures to the right. Comparing the results for each function, the Pareto curves and points look identical. Also, the MCACO Pareto optimal points are evenly spread out which is a noteworthy result.

5.2. Benefits and Advantages

A few benefits of the MCACO algorithm are discussed next. The ants in the MCACO algorithm are able to thoroughly explore the search domain. They move in

many possible directions around the domain to areas of better fitness and generally find the global minimum resulting in a very accurate algorithm. Also, a few of the functions tested had global minimums at several different locations. The MCACO algorithm has the potential ability to find each of the global minimums in one run. The ants move around the domain and continuously locate areas of better fitness. Along the way to the minimum, ants may pass and search around many local minimums, some which are global minimums. For example, consider the Branin function. The function has three global minimums located at the following locations:

$$(\pi, 2.275),\ (-\pi, 12.275),\ \text{and}\ (9.42478, 2.475) \qquad (50)$$

The following figure shows the complete ant movements for the Branin function and the locations of the global minimums for one complete run of the algorithm:

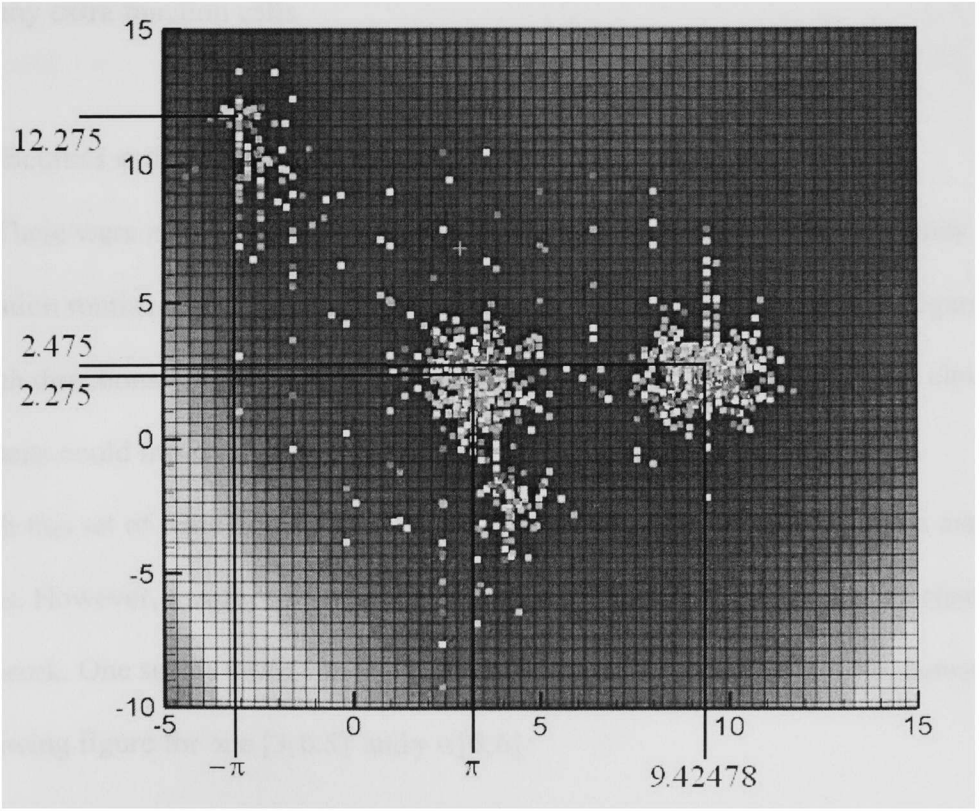


Figure 69: Branin function ants and minimums

Notice that the ants found each of the areas that contained the global minimum and searched around each them.

Similarly, when the ant movements and locations are plotted on graphs, the general areas of minimum fitness can be identified. Examples of this are shown in the appendix. Another advantage is the versatility of the algorithm. If accuracy of the result is the primary concern and not computing time, the precision of the MCACO algorithm can be further increased. This can be accomplished by shrinking the radius slower and by having more initial starting nests. At the expense of function calls, the results become even more accurate. Likewise, the opposite is true when a lower accuracy is acceptable which would result in a smaller amount of function calls. The number of function calls is also independent of function topology, which means that difficult functions do not require any extra function calls.

5.3. Difficulties and Limitations

There were many difficulties in building the modified continuous ant colony optimization routine. One of the first difficulties encountered was the decision regarding the search directions the ants could travel in. Initially, there were four directions chosen that the ants could travel in; they were situated at 45, 135, 225, and 315 degrees. Although this set of search direction is rather limited, the ants performed well on many functions. However, there were a few functions for which this search direction scheme did not work. One such example is the Rosen function. The Rosen function is shown in the following figure for $x \in [3, 6.5]$ and $y \in [0, 6]$:

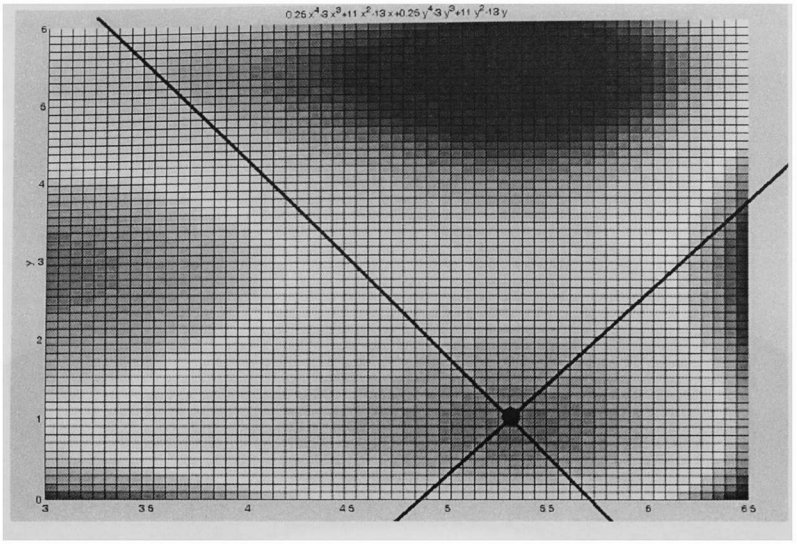


Figure 70: Rosen function in xy plane

Occasionally, an ant would get stuck at point (5.33,0.87), designated by the black dot. The ant would have no chance of escaping in the four degrees previously mentioned because the global minimum is directly overhead. This problem was combated by initially switching between two sets of directions and then using randomized search directions.

Difficulties were also met when applying the NNC method. Take into consideration the Rendon multi-objective function. Following the steps of the method as outlined in the methodology chapter, the following first constraint is developed,

$$\left(\left(\left(\frac{1}{x^2+y^2+1}\right)-0\right)-\left(\left(\frac{x^2+3*y^2+1}{37}\right)-1\right)\right)\leq 0 \qquad (51)$$

This constraint would be the result of case when,

$$\alpha_{1j}=0 \quad \text{and} \quad \alpha_{2j}=1 \qquad (52)$$

The following figures show the constraint in graphic form:

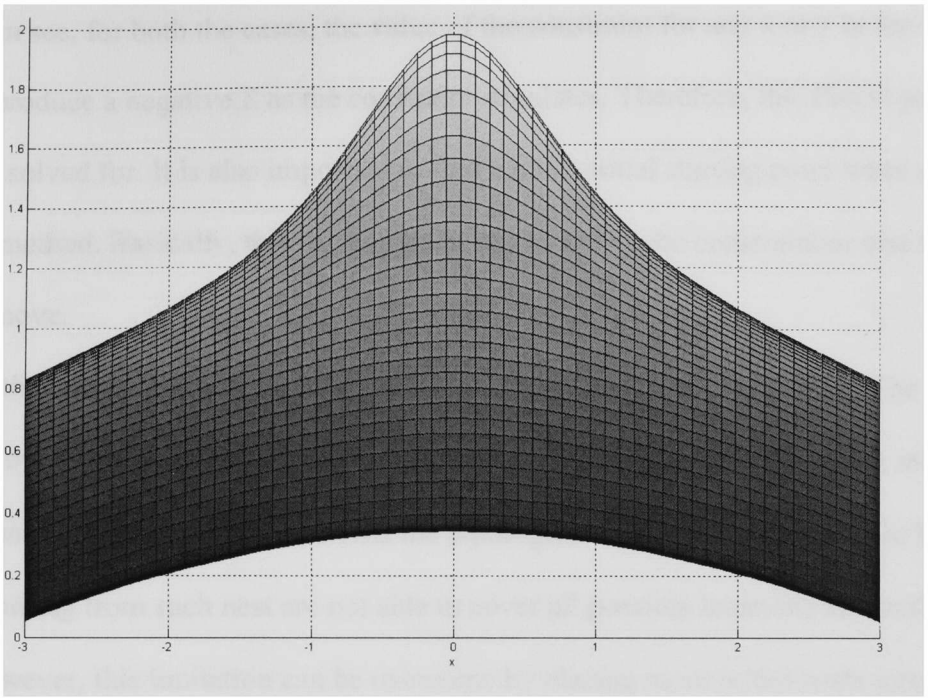


Figure 71: Rendon function xz view

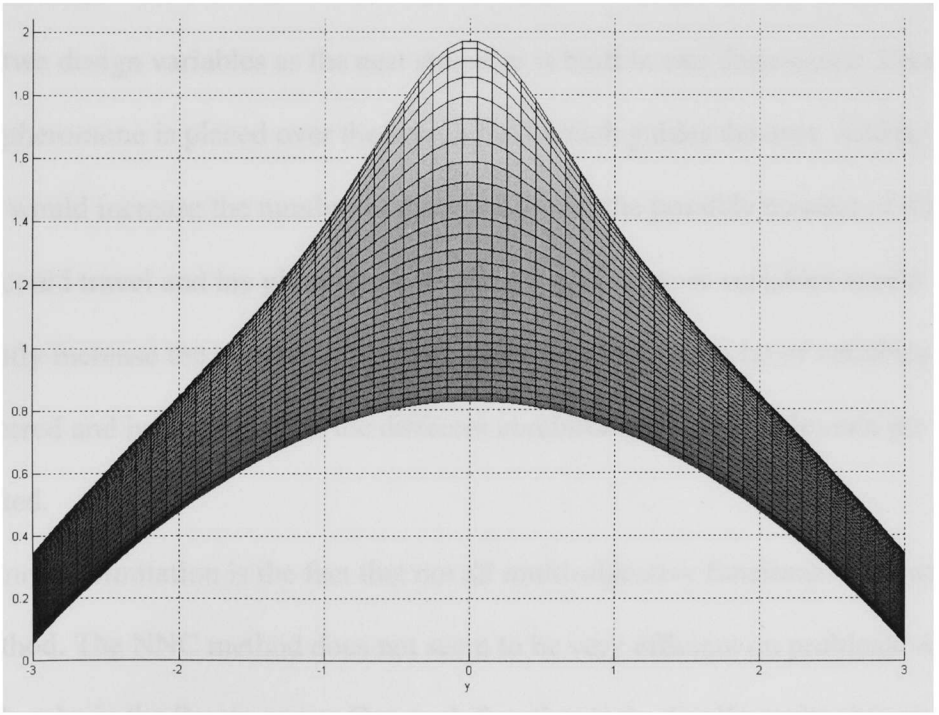


Figure 72: Rendon function yz view

As you can see, for both the cases, the value of the constraint for any x or y in the domain does not produce a negative Z as the constraint stipulates. Therefore, this Pareto point cannot be solved for. It is also important to find a good initial starting point when using the NNC method. Basically, this starting point has to satisfy the constraint or else the ant does not move.

Although robust in a sense, MCACO does have numerous limitations. The algorithm only works over a limited domain range. This is because only sixteen initial nests are used and must be placed across the topological space. If the space is too large, the ants coming from each nest are not able to cover all possible locations across the space. However, this limitation can be overcome by placing more initial nests across the domain at the expense of an increased number of function calls. Another limitation is the number of design variables that MCACO can handle. Currently, the algorithm only supports two design variables as the nest structure is built in two dimensions. Recall that artificial pheromone is placed over the search field which guides the ants. Adding more variables would increase the number of dimensions and the possible number of directions that ants could travel and lay pheromone in. Hence, adding more variables would significantly increase the number of function calls. MCACO has a lot of variables that can be altered and balancing all of the different combinations of variables can get complicated.

Another limitation is the fact that not all multi-objective functions work with the NNC method. The NNC method does not seem to be very efficient on problems with multiple breaks in the Pareto curve. One such function is the Coello multi-objective function. The algorithm was attempted on the Coello function but the results were not up

to par. In its current state, the number of function calls for MCACO is only moderately competitive with other optimization algorithms. Also, functions for which the difference between a global minimum and a local minimum is very small, have the possibility to confuse the ants. For example, the Perm #2 function has the following global and local minimum:

$$\begin{aligned} &\text{global min is } f(x,\ y) = 0 \text{ located at } (x,\ y) = (1,\ 0.5) \\ &\text{local min is } f(x,\ y) = 0.000205 \text{ located at } (x,\ y) = (0.4945,\ 0.9955) \end{aligned} \tag{53}$$

Occasionally, the ants might find the local minimum and consider it to be the global minimum because the value at the local minimum might be slightly lower than the value at the global minimum. The ants however, do find the global minimum as well because they explore the area around it.

CHAPTER 6

CONCLUSION

6.1. Recommendations for Future Research

There are a few improvements and modifications that can be researched to enhance the overall capacity and efficiency of the algorithm. First of all, 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. Another interesting suggestion would be to try running the MCACO algorithm with the NNC method for more than two objectives functions. The steps for using the NNC method for a general n-objective are similar to those of the bi-objective case [25]. The problem that would need to be solved for each generated $\bar{X}_{pj}$ point is shown as follows [25],

$$
\begin{aligned}
&\text{Problem P3 for } j^{th} \text{ point} \\
&\min_{x} \ \bar{\mu}_n \\
&\text{subject to:} \\
&g_j(x) \le 0, \quad (1 \le j \le r) \\
&h_k(x) = 0, \quad (1 \le k \le s) \\
&x_{li} \le x_i \le x_{ui}, \quad (1 \le i \le n_x) \\
&\bar{N}_k(\bar{\mu} - \bar{X}_{pj})^T \le 0, \quad (1 \le k \le n-1) \\
&\bar{\mu} = \left[\bar{\mu}_1(x), \ldots, \ \bar{\mu}_n(x)\right]^T
\end{aligned}
\tag{54}
$$

Essentially, we would end up with n constraints that would be applied to each point on the utopia hyperplane [25].

Values inside the MCACO algorithm could also be tinkered with to see if better

all around results can be obtained. 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 mores 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. Another consideration would be to try to working on the previous version of the single-objective MCACO algorithm that contains a single nest. The results from that version of the algorithm are given as follows:

Table 30: Previous version MCACO results

Function name	Average number of function calls for 50 runs	Standard deviation of function calls	Success rate %
Bohachevsky	562	43	70
Booth	468	49	100
Branin	562	24	96
Easom	350	24	80
Goldstein and Price	472	59	76
Hump	503	41	94
Rastrigin	554	39	52
Rosen	388	40	66

The number of function calls and the success rates for some of the easier functions, such as Booth and Branin, are excellent. However, when the more difficult optimization test

cases are tried, the method becomes unstable and the success rate drops. The idea contained in this version of the MCACO algorithm may be researched further to raise the success rate while keeping the number of function calls relatively low. 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.

6.2. 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 minimums obtained and their respective locations, is very small. Accuracy is very good for the MCACO method because the MCACO results indicate that the minimums obtained for the test cases closely resemble the true analytic minimums. Although it is not yet comparable with other top-tier optimization methods in terms of function calls, there is still room for improvement.

LIST OF REFERENCES

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

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

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

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

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

[6] M. Matsumoto and T. Nishimura, "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, 1998.

[7] A. Antoniou and W. Lu, *Practical Optimization: Algorithms and Engineering Applications*, New York, Springer, 2007.

[8] S. Rao, *Engineering Optimization: Theory and Practice, 3rd Edition*, New York, Wiley-Interscience, 1996.

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

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

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

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

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

[14] M. Dorigo and K. Socha, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 158, no. 3, pp. 1155-1173, 2008.

[15] G. Bilchev and I. Parmee, "Constrained Optimization With An Ant Colony Search Model," *Proceedings of ACEDC'96*, Plymouth, UK, pp. 141-151, 1996.

[16] J. Dreo and P. Siarry, "Continuous interacting ant colony algorithm based on dense heterarchy," *Future Generation Computer Systems*, vol. 20, no. 5, pp. 841-856, 2004.

[17] M. Kong and P. Tian, "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, 2006.

[18] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, New York, Wiley, 2001.

[19] R. Marler and J. Arora, "Survey of Multi-Objective Optimization Methods for Engineering," *Structural and Multidisciplinary Optimization,* vol. 26, no. 6, pp. 369-395, 2004.

[20] K. Miettinen, *Nonlinear Multiobjective Optimization*, New York, Springer, 1998.

[21] Y. Donoso and R. Fabregat, *Multi-Objective Optimization in Computer Networks Using Metaheuristics*, Chicago, Auerbach, 2007.

[22] I. Das and J. Dennis, "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, 1997.

[23] A. Messac, G. Sundararaj, R. Tappeta, and J. Renaud, "Ability of Objective Functions to Generate Points on Non-Convex Pareto Frontiers," *AIAA Journal*, Vol. 38, No. 6, pp. 1084-1091, 2000.

[24] A. Messac and C. Mattson, "Normal Constraint Method with Guarantee of Even Representation of Complete Pareto Frontier," *AIAA Journal*, Vol. 42, No. 10, pp. 2101-2111, 2004.

[25] A. Messac, A. Ismail-Yahaya, and C. Mattson, "The Normalized Normal Constraint Method for Generating the Pareto Frontier," *Structural and Multidisciplinary Optimization*, Vol. 25, No. 2, pp. 86-98, 2003.

[26] M. Martinez, J. Sanchis, and X. Blasco, "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, 2007.

[27] J. Sanchis, M. Martinez, X. Blasco, and J. Salcedo, "A New Perspective on Multiobjective Optimization by Enhanced Normalized Normal Constraint Method," *Structural and Multidisciplinary Optimization*, Springer Berlin, 2007.

[28] M. Matsumoto and T. Nishimura, "Dynamic Creation of Pseudorandom Number Generators," *Monte Carlo and Quasi-Monte Carlo Methods 1998*, Springer, pp. 56-69, 2000.

[29] J. Kapur and H. Saxena, *Mathematical Statistics,* New Delhi, S. Chand, 1997.

[30] I. Egorov, "IOSO NM Version 1," User Guide, IOSO Technology Center, 2003.

[31] M. Kong and P. Tian, "A Binary Ant Colony Optimization for the Unconstrained Function Optimization Problem," *Lecture Notes in Computer Science*, Vol. 3801, pp. 682-687, 2005.

[32] M. Kong and P. Tian, "A Direct Application of Ant Colony Optimization to Function Optimization Problem in Continuous Domain," *Lecture Notes in Computer Science*, Vol. 4150, pp. 324-331, 2006.

[33] K. Socha, "ACO for Continuous and Mixed-Variable Optimization," *Lecture Notes in Computer Science*, Vol. 3172, pp. 25-36, 2004.

APPENDICES

Appendix A: Sample Ant Movements

The following figures show the ant movement over the domain with the grayscale plot in the background. Note that the darker the color, the better the function fitness is in terms of the minimum. If the figures did not have the function topology as the background, they would still prove to be insightful because they would show the general areas of minimum function values across the topology.

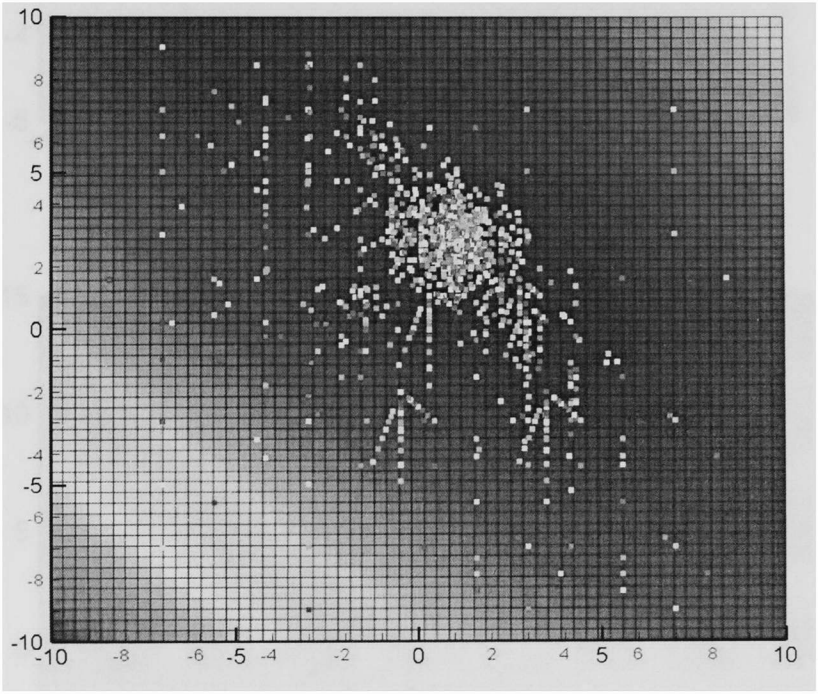


Figure 73: Booth function ant movement

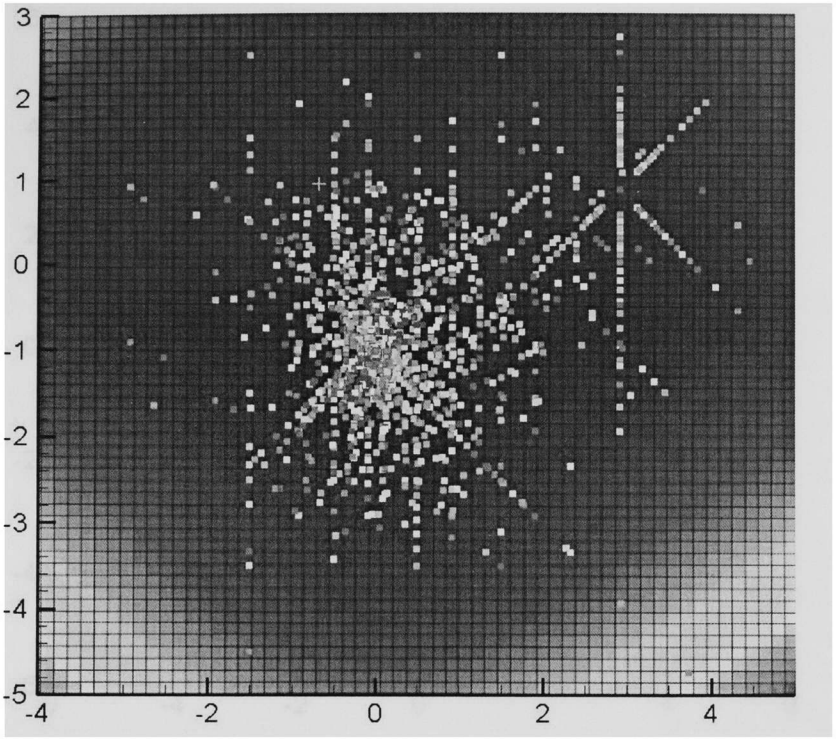


Figure 74: Goldstein and Price function ant movement

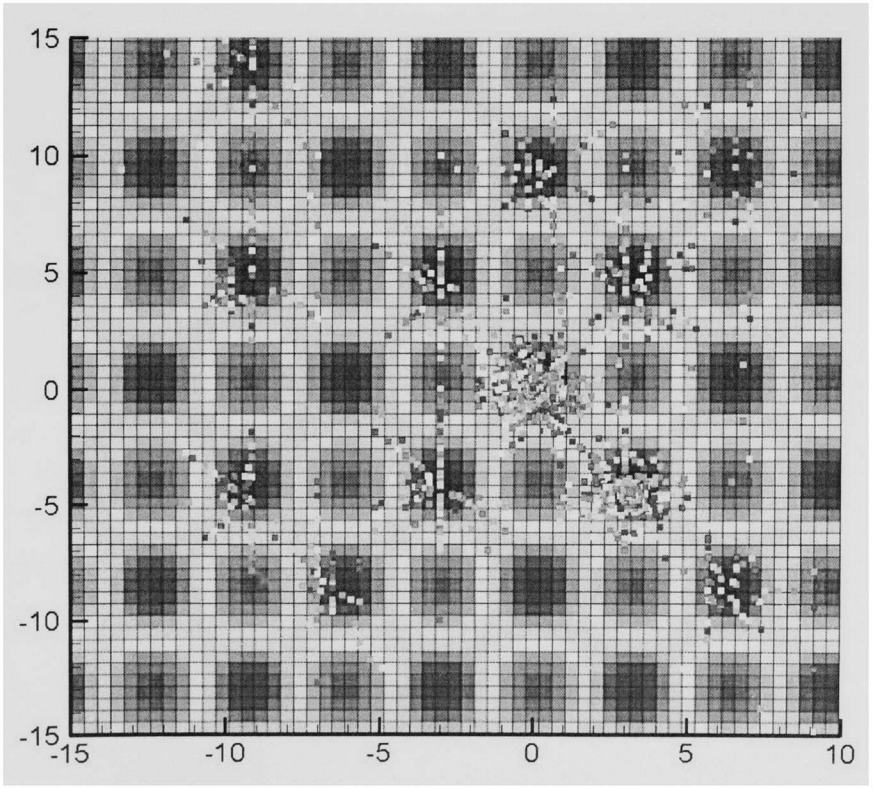


Figure 75: Griewank function ant movement

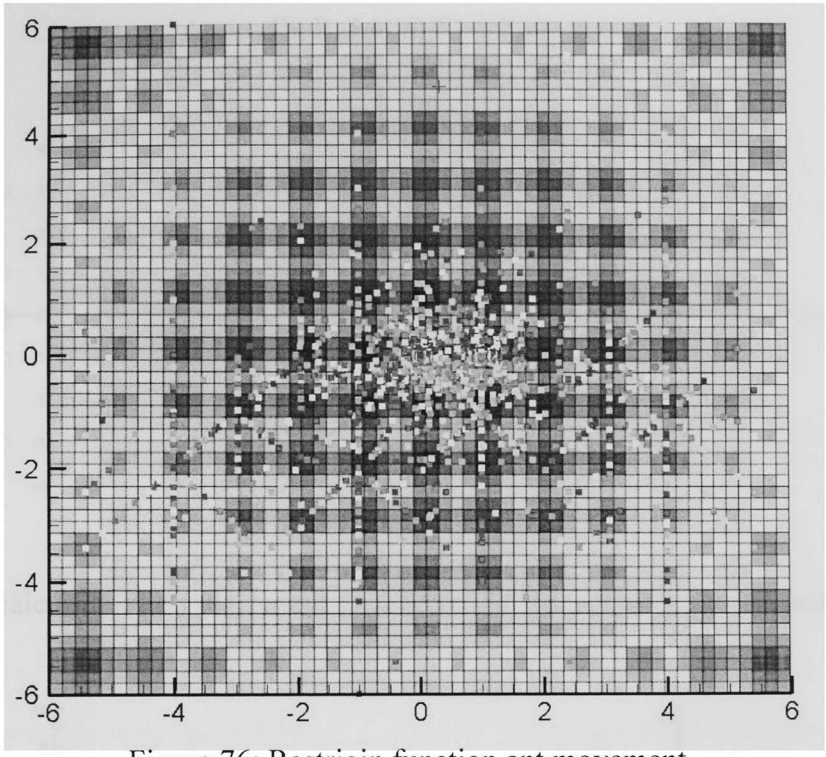


Figure 76: Rastrigin function ant movement

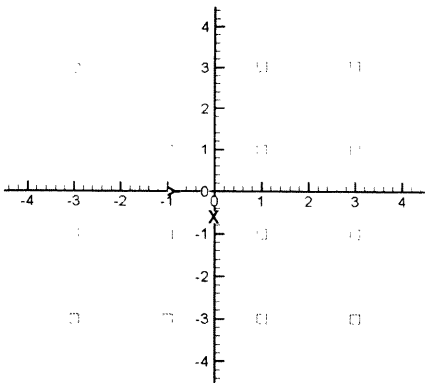


Figure 77: Beale initial nest placement

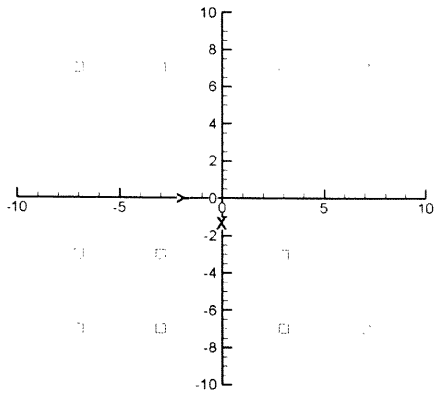


Figure 78: Bohachevsky initial nest placement

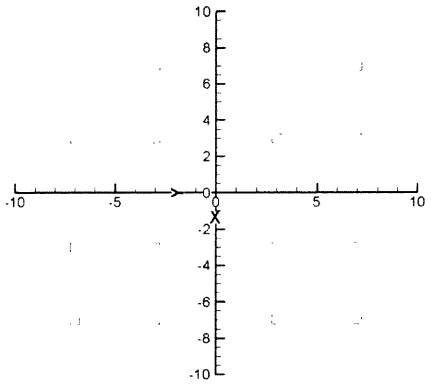


Figure 79: Booth initial nest placement

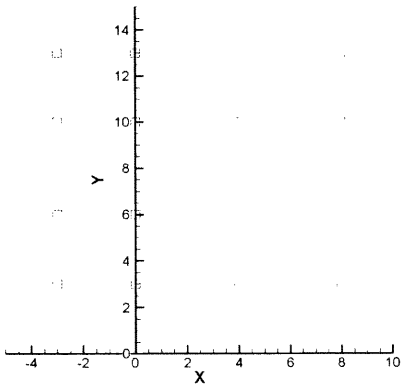


Figure 80: Branin initial nest placement

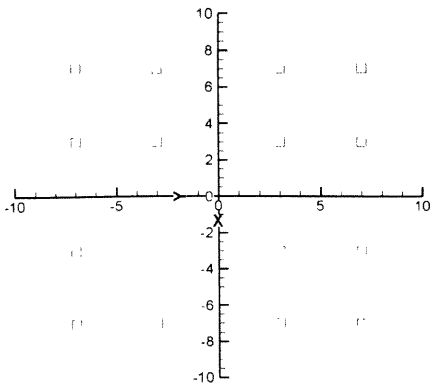


Figure 81: Easom initial nest placement

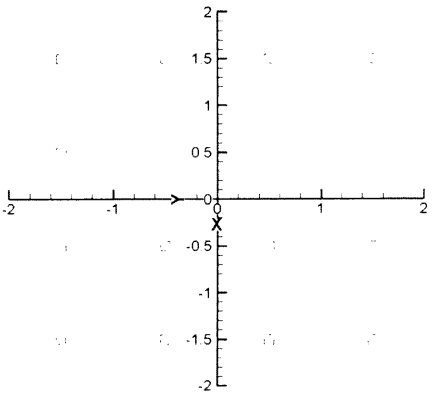


Figure 82: GP initial nest placement

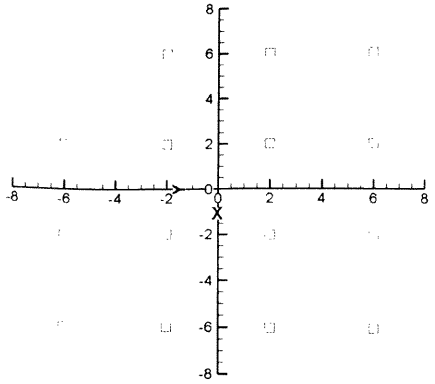


Figure 83: FR initial nest placement

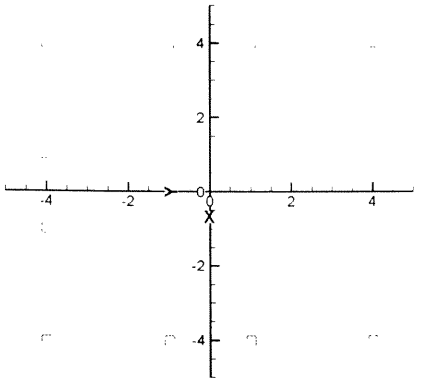


Figure 84: Hump initial nest placement

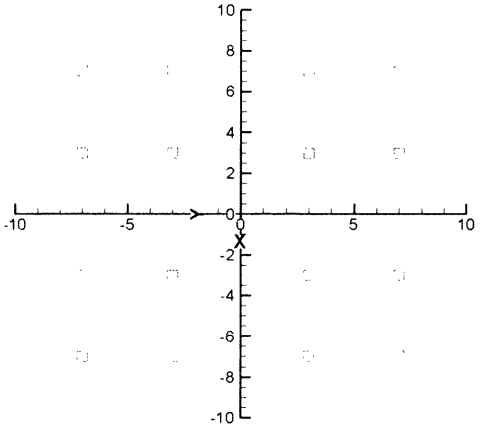


Figure 85: Griewank initial nest placement

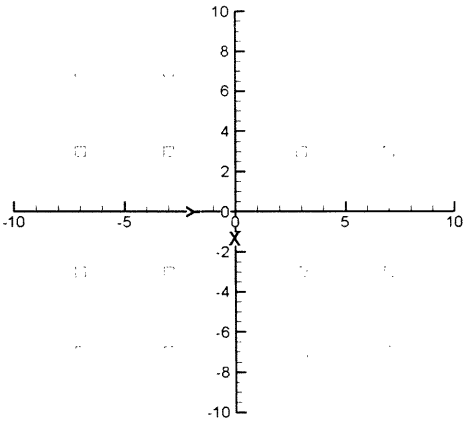


Figure 86: Matyas initial nest placement

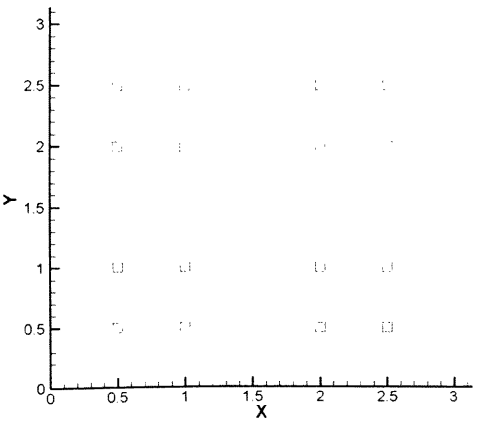


Figure 87: Michalewics initial nest placement

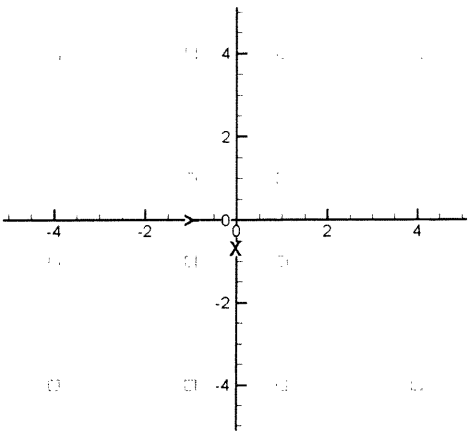


Figure 88: Rastrigin initial nest placement

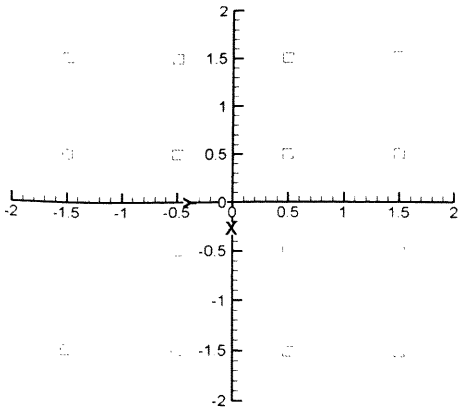


Figure 89: Rosenbrock initial nest placement

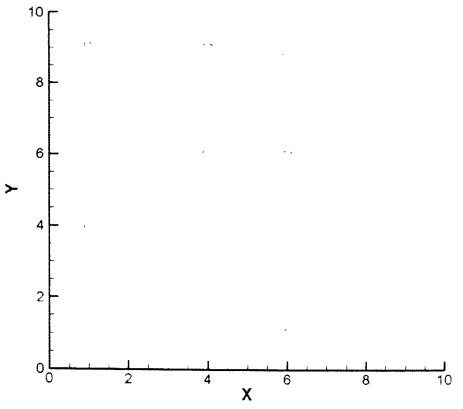


Figure 90: MG initial nest placement

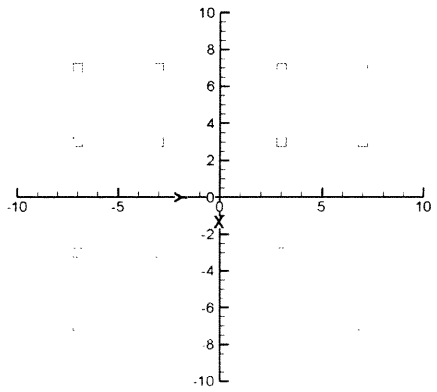


Figure 91: Shubert initial nest placement

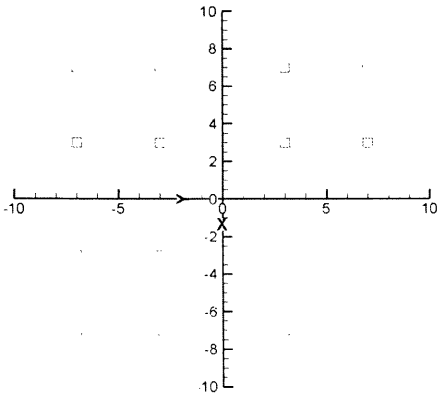


Figure 92: Rosen initial nest placement

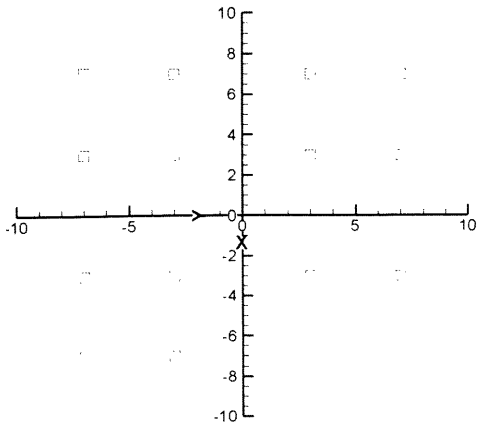


Figure 93: Ackley initial nest placement

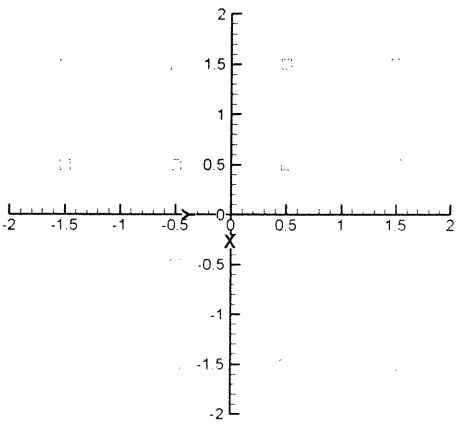


Figure 94: Perm # 1 initial nest placement

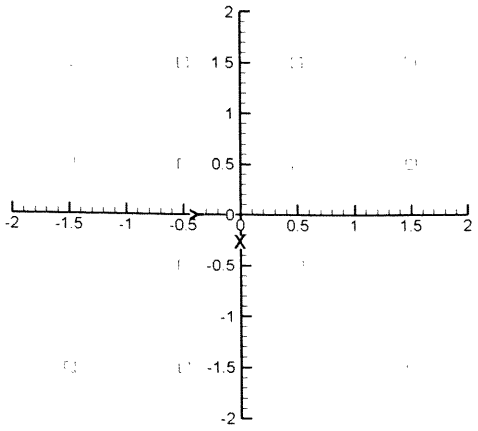


Figure 95: Perm #2 initial nest placement

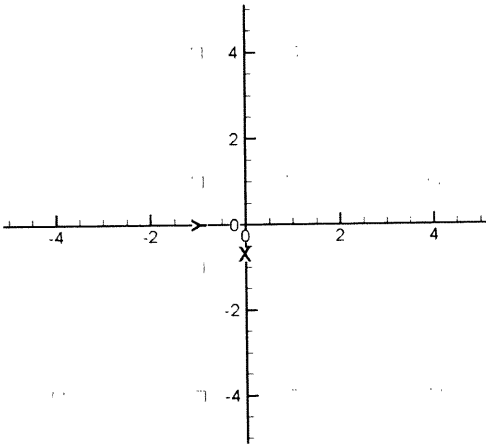


Figure 96: Sphere initial nest placement