Bacteria Foraging Optimization with Genetic Operators for QAP and mQAP

Saeid Parvandeh

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
July 2013
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammad Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Asst. Prof. Dr. Ahmet Ünveren
Supervisor

Examining committee

1. Asst. Prof. Dr. Adnan Acan

2. Asst. Prof. Dr. Önsen Toygar

3. Asst. Prof. Dr. Ahmet Ünveren

ABSRACT

The Bacterial Foraging Optimization Algorithm (BFOA) is one of the metaheuristics algorithms which is widely used in Optimization processes. It is also related to other optimization algorithms such as Ant Colony and Particle Swarm Optimization. So far, many of the metaheuristics algorithms such as genetic algorithm, particle swarm optimization, and tabu search have been used to hybridize this algorithm.

The BFOA is imitated by behavior of the foraging bacteria group such as E.coli. Basically, the main aim of the algorithm is to eliminate those bacteria which have weak foraging methods and following up those bacteria which have strong foraging methods. In this extent, each bacterium contacts to other bacteria by sending signals such that bacterium change the position to the next step if prior factors have been satisfied. In fact, the process of algorithm allows bacteria to follow up the nutrients toward the optimal. BFO algorithm has three steps: 1) 'Chemo-tactic', 2) 'Reproduction', and 3) 'Elimination-dispersal'.

In this thesis, Bacteria Foraging Optimization Algorithm (BFOA) is used for the solution of Quadratic Assignment Problems (QAP), and Multi-objective QAP (mQAP). Since, QAP is NP-hard problem and finding a reasonable solution is a non-polynomially time-consuming process, then one of the combinatorial algorithms should be used to find the solution in reasonable time. The BFO is one of the combinatorial optimization algorithms which apply to optimize the cost of such problems. The BFO algorithm takes a population of permutation (locations), and in several iterations of a generation, it can find a reasonable solution for QAP or mQAP. Furthermore, in order to improve the algorithm, some genetic updating operators such as crossover and mutation have been used in the second part of BFOA algorithm (chemo-tactic) in every generation of this step. Additionally, robust tabu search has been used in the third part (elimination-dispersal) to improve the best solution found so far.

Keywords: BFOA, Tabu Search, Quadratic Assignment Problem, Multiobjective Quadratic Assignment Problem

ÖZ

Bakteriyel besin arama algoritması (BBAA), bakteri eniyleme optimizyasyon ve sürü en iyileme algoritmaları alanına ait olup daha geniş alanlar olan hesaplara dayalı zeka ve sezgisel Algoritma alanları altında kullanılan noktadır. BBAA algoritması parçası sürü eniyileme ve karınca kolonisi en iyileme algoritmaları ilede benzerlik göstermektedir. Ayrıca BBAA, diğer en iyileme algoritmaları ile birleştirilerek de kullanılmaktadır. Örneğin BBAA ve Genetik algoritma, veya parçacık sürü en iyileme algoritması veya Tabu arama algoritmaları.

BBAA, E.coli bakterisinin beslenme davranışından esinlenerek geliştirilmiş bir iyileme yöntemidir. E.coli bakterisi besin maddesine ulaştığında diğer bakterileri uyarıcı etkiye sahip kimyasal bir madde salgılamaktadır. Bu madde diğer E.coli bakterilerinin besini bulan bakterinin bulunduğu yere doğru hareket etmesini sağlamaktadır. BBAA en iyileme algoritması tüm hücrelerin en iyiye doğru grup halinde hareketini sağlama stratejisi gütmektedir.

En iyiye ulaşmak için BBAA sırasıyla üç önemli işlemi sürü halindeki tüm hücrelere uygular; 1) “Kemotaktik”, 2) “Üreme”, 3) “Eliminasyon-dağıtım”.

Bu tezde bakteriyel besin arama algoritması (BBAA) kullanılarak karesel atama problemleri(KAP) ve çok amaçlı karesel atama problemleri (MKAP) çözülmüştür.

KAP, bir dizi aracı, bir dizi lokasyona, verilen lokasyonlar arası uzaklıklar ve araçlar arası akış bilgileri kullanarak atama yapma problemi olarak tanımlıya biliriz. KAP problemi bir NP-ZOR problem olduğundan iyi çözümlere ulaşmak uzun zaman almaktadır. BBAA algoritması tümleşik algoritmalardan biri olup NP-ZOR problemlerini çözmek için kullanılır.

BBAA algoritması en iyileme döngüsü ile KAP ve MKAP problemlerine çözüm bulmaya çalışır. BBAA algoritmasında KAP ve MKAP problemleri çözümü esnasında farklı çaprazlama ve mutasyon metodları kullanılarak mevcut problemler iyileştirilmeye çalışılmıştır. Ayrıca Tabu arama algoritması bulunan en iyi çözümlere uygulanmış ve mevcut en iyi çözümlere ulaşılmıştır.

Anahtar Kelimeler: BBAA , Tabu Arama, Karesel Atama Problemleri, Çok Amaçlı Karesel Atama Problemleri

I wish I could write peace of note to my girlfriend to compensate her spirit of giving to me in every moment, but this paper and keyboard are not able to keep any note to countervail her kindly accompaniment during two years. I would like to dedicate this thesis to her.

ACKNOWLEDGMENTS

I would like to thank Asst. Prof. Dr. Ahmet Ünveren for his continuous support and guidance in the preparation of this study. He has always been available to answer my questions so kindly. Without his invaluable supervision, all my efforts could have been short-sighted.

I am also obliged to Asst. Prof. Dr. Adan Acan for his help during my thesis. He has helped me to think about problems in deep with his quite comprehension exams and assignments. Moreover, I really appreciate my friends who have always been around to support me morally. I would like to thank them also. I owe quit a lot to my family who allowed me to study abroad. I would like to appreciate them and I wish to satisfy them in the future life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) is one of the reputed NP-hard combinatorial optimization problems such that there is no known polynomial time algorithm for its solution. The QAP has been introduced by Koopmans and Beckmann in 1957 [1]. It can be defined as a problem to allocate a facility set to a location set with mutual distances among the locations and mutual flow among the facilities. Specially, given two $n \times n$ matrices $A = \left(f_{ij}\right)$ and $B = (d_{kl})$ as input, in which all the elements are real, and let f_{ij} be flow among facility i and facility j, d_{kl} be the distance among the location k and location l. Also, let n be the number of facilities and locations, where $n = \{1, 2, 3, \dots, n\}$. The formulation of the QAP can be defined as follows [1]:

$$\underset{\varphi \epsilon S_n}{min} Q(\varphi) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\varphi(i)\varphi(j)} \qquad (1)$$

where, S_n is the set of all permutations of n locations. Each individual product $f_{ij} d_{\varphi(i)\varphi(j)}$ computes the cost of assigning facility i to location $\varphi(i)$ and facility j to location $\varphi(j)$. An QAP instance with input matrices *A* and *B* is denoted by QAP (*A*, *B*). It is to be noted that the number of facilities (*n*) is assumed to be the same as the number of locations. In other words, one facility could be assigned to only one location, and one location could be assigned to only one facility in a feasible assignment solution, otherwise it is infeasible assignment solution [3, 4].

The QAP problem can be defined as follows

$A = \left(f_{ij}\right)$ = matrix of the flows (between facility *i* and facility *j*);

$B = (d_{kl})$ = matrix of distances (between location *k* and location *l*).

After this construction, a permutation $\varphi: i \to \varphi(i)$ can be introduced as a particular assignment of facility $i = \varphi(i)$ to location $j\ (j = 1,2,3, \dots n)$. The cost of transferring data between two facilities can be expressed as the product of the distances between the locations to which the facilities are assigned by the flow between the two facilities, $f_{ij} d_{\varphi(i)\varphi(j)}$. In order to solve the QAP, a permutation φ of the indices *1,2,3,...,n* which minimizes the local assignment cost should be found.

Here is an example of QAP:

Given the following flow and distance matrices:

The flow matrix $F = (f_{ij})_{4\times4}$between facilities is given as:

$$F = (f_{ij})_{4\times4} = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 4 \\ 2 & 1 & 4 & 0 \end{bmatrix}$$

and the distance matrix $D = (d_{kl})_{4\times4}$between locations is specified as:

$$D = (d_{kl})_{4\times4} = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{bmatrix} = \begin{bmatrix} 0 & 22 & 53 & 0 \\ 22 & 0 & 0 & 0 \\ 53 & 40 & 0 & 55 \\ 0 & 0 & 55 & 0 \end{bmatrix}$$

In the list below, *24* solutions for the above QAP has been shown. In fact, each of the arrangements is a permutation of four facilities to be assigned to four locations.

1.(1,2,3,4)	7. (3,4,2,1)	13. (4,3,2,1)	19. (1,2,4,3)
2.(4,1,2,3)	8. (1,3,4,2)	14. (3,2,1,4)	20. (2,4,3,1)
3.(3,4,1,2)	9. (1,3,2,4)	15. (2,1,4,3)	21. (4,2,3,1)
4.(2,3,4,1)	10. (4,1,3,2)	16. (1,4,3,2)	22. (2,3,1,4)
5.(2,1,3,4)	11. (2,4,1,3)	17. (4,3,1,2)	23. (3,1,4,2)
6.(4,2,1,3)	12. (3,2,4,1)	18. (3,1,2,4)	24. (1,4,2,3)

Let's select permutation *15* randomly. In this permutation, four facilities have been arranged in this order: (2, 1, 4, 3), in which facility 2 is to be assigned in location *1*, facility *1* to be assigned in location 2, and so on. Figure 1 shows the graph model of this permutation.

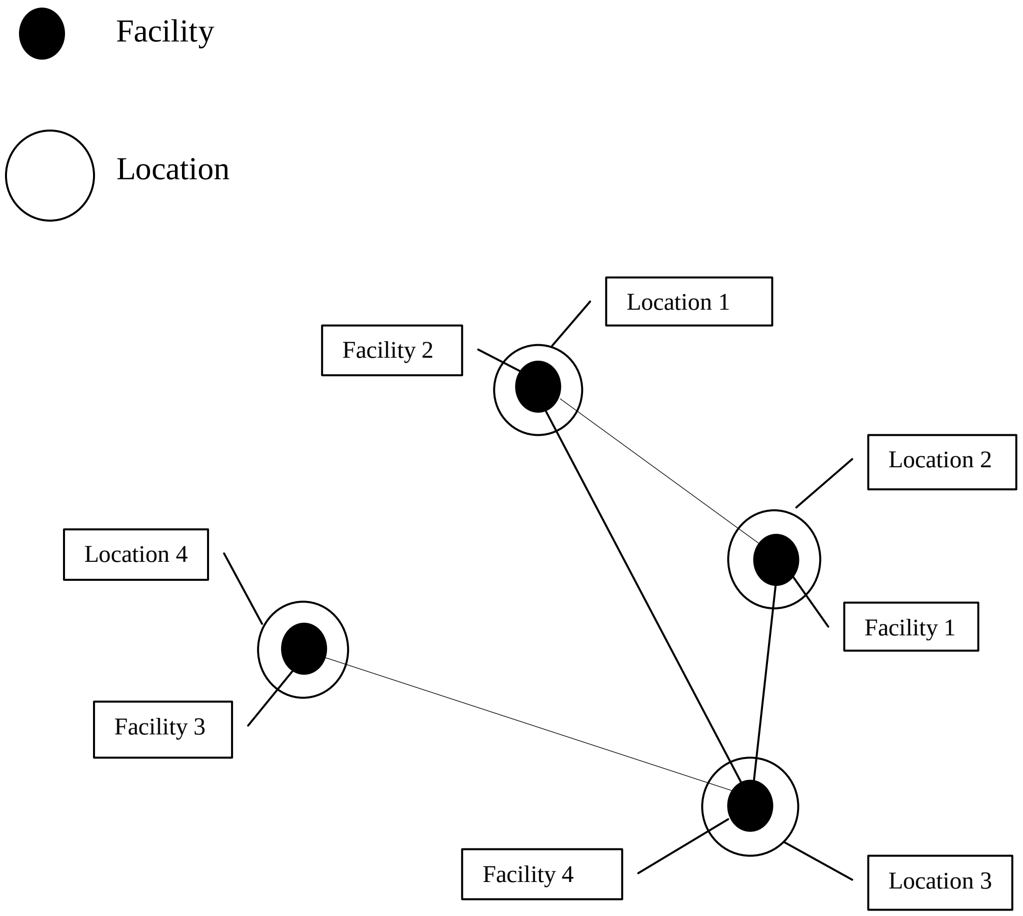


Figure 1: A Solution of the Example QAP Instance

Therefore, the cost of this assignment can be computed as follows:

$$Q(2,1,4,3) = (f_{11} \times d_{p(1)p(1)} + f_{12} \times d_{p(1)p(2)} + f_{13} \times d_{p(1)p(3)} + f_{14} \times d_{p(1)p(4)} + f_{21} \times d_{p(2)p(1)} f_{22} \times d_{p(2)p(2)} + f_{23} \times d_{p(2)p(3)} + f_{24} \times d_{p(2)p(4)} + f_{31} \times d_{p(3)p(1)} + f_{32} \times d_{p(3)p(2)} + f_{33} \times d_{p(3)p(3)} + f_{34} \times d_{p(3)p(4)} + f_{41} \times d_{p(4)p(1)} + f_{42} \times d_{p(4)p(2)} + f_{43} \times d_{p(4)p(3)} + f_{44} \times d_{p(4)p(4)})$$

$$= (f_{11} \times d_{22} + f_{12} \times d_{21} + f_{13} \times d_{24} + f_{14} \times d_{23} + f_{21} \times d_{12} + f_{22} \times d_{11} + f_{23} \times d_{14} + f_{24} \times d_{13} + f_{31} \times d_{42} + f_{32} \times d_{41} + f_{33} \times d_{44} + f_{34} \times d_{43} + f_{41} \times d_{32} + f_{42} \times d_{31} + f_{43} \times d_{34} + f_{44} \times d_{33})$$

$= 3 \times 22 + 0 \times 0 + 0 \times 0 + 1 \times 53 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 4 \times 55 + 2 \times 40 + 1 \times 53 + 4 \times 55 + 0 \times 0) = 838$

The calculation of the cost of one solution for facility/location assignment has been shown above.

1.2 Multiobjective Quadratic Assignment Problem

Knowles and Corne introduced the other QAP version as multiobjective QAP (mQAP) [2]. In this case, the mQAP has multiple flow matrices and a distance matrix. The mQAP is more commonly used where one facility should be assigned to one location with respect to the multiple flow matrices and with a distance matrix such that flow matrices are different to each other. So, the mQAP can be modeled as follows [2]:

$$\underset{\varphi \epsilon S_n}{min} \bar{Q}(\varphi) = \{Q^1(\varphi), Q^2(\varphi), \cdots, Q^m(\varphi)\} \tag{2}$$

where,

$$Q^p(\varphi) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}^p d_{\varphi(i)\varphi(j),} \qquad 1 \le p \le m \tag{3}$$

In this formula, the f_{ij}^p indicates p^{th} flow between facility i and facility j, and m is the number of objectives and "min" means to obtain the Pareto front [19]. Here is an example of mQAP:

Assuming previous example for QAP, here, instead of one flow matrix, there are two flow matrices in the following formats:

$$F_1 = (f_{ij})_{4\times4} = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 4 \\ 2 & 1 & 4 & 0 \end{bmatrix}$$

$$F_2 = (f_{ij})_{4\times4} = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 4 \\ 3 & 2 & 4 & 0 \end{bmatrix}$$

$$D = (d_{kl})_{4\times4} = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{bmatrix} = \begin{bmatrix} 0 & 22 & 53 & 0 \\ 22 & 0 & 0 & 0 \\ 53 & 40 & 0 & 55 \\ 0 & 0 & 55 & 0 \end{bmatrix}$$

Here, the QAP formula should be calculated for two flow matrices with the same distance matrix. That means there are two solutions according to flows between facilities. So, by choosing one of the feasible permutations as random, the numerical calculation of mQAP according to Equations (2) and (3) can be calculated as follows:

$$Q^1(2,1,4,3) = (f_{11} \times d_{p(1)p(1)} + f_{12} \times d_{p(1)p(2)} + f_{13} \times d_{p(1)p(3)} + f_{14} \times d_{p(1)p(4)} + f_{21} \times d_{p(2)p(1)} + f_{22} \times d_{p(2)p(2)} + f_{23} \times d_{p(2)p(3)} + f_{24} \times d_{p(2)p(4)} + f_{31} \times d_{p(3)p(1)} + f_{32} \times d_{p(3)p(2)} + f_{33} \times d_{p(3)p(3)} + f_{34} \times d_{p(3)p(4)} + f_{41} \times d_{p(4)p(1)} + f_{42} \times d_{p(4)p(2)} + f_{43} \times d_{p(4)p(3)} + f_{44} \times d_{p(4)p(4)})$$

$= (f_{11} \times d_{22} + f_{12} \times d_{21} + f_{13} \times d_{24} + f_{14} \times d_{23} + f_{21} \times d_{12} + f_{22} \times d_{11} + f_{23} \times d_{14} + f_{24} \times d_{13} + f_{31} \times d_{42} + f_{32} \times d_{41} + f_{33} \times d_{44} + f_{34} \times d_{43} + f_{41} \times d_{32} + f_{42} \times d_{31} + f_{43} \times d_{34} + f_{44} \times d_{33})$

$= (0 \times 0 + 3 \times 22 + 0 \times 0 + 2 \times 40 + 3 \times 22 + 0 \times 0 + 0 \times 0 + 1 \times 53 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 4 \times 55 + 2 \times 40 + 1 \times 53 + 4 \times 55 + 0 \times 0) = 838$

$Q^2(2,1,4,3) = (f_{11} \times d_{p(1)p(1)} + f_{12} \times d_{p(1)p(2)} + f_{13} \times d_{p(1)p(3)} + f_{14} \times d_{p(1)p(4)} + f_{21} \times d_{p(2)p(1)} + f_{22} \times d_{p(2)p(2)} + f_{23} \times d_{p(2)p(3)} + f_{24} \times d_{p(2)p(4)} + f_{31} \times d_{p(3)p(1)} + f_{32} \times d_{p(3)p(2)} + f_{33} \times d_{p(3)p(3)} + f_{34} \times d_{p(3)p(4)} + f_{41} \times d_{p(4)p(1)} + f_{42} \times d_{p(4)p(2)} + f_{43} \times d_{p(4)p(3)} + f_{44} \times d_{p(4)p(4)})$

$= (f\_11 \times d\_22 + f\_12 \times d\_21 + f\_13 \times d\_24 + f\_14 \times d\_23 + f\_21 \times d\_12 + f\_22 \times d\_11 + f\_23 \times d\_14 + f\_24 \times d\_13 + f\_31 \times d\_42 + f\_32 \times d\_41 + f\_33 \times d\_44 + f\_34 \times d\_43 + f\_41 \times d\_32 + f\_42 \times d\_31 + f\_43 \times d\_34 + f\_44 \times d\_33)$

$= (0 \times 0 + 1 \times 22 + 0 \times 0 + 3 \times 0 + 1 \times 22 + 0 \times 0 + 0 \times 0 + 2 \times 53 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 4 \times 55 + 3 \times 40 + 2 \times 53 + 4 \times 55 + 0 \times 0) = 830$

Based on the above calculations, since there are two flow matrices, two objectives will be obtained accordingly.

1.3 Dominance and Pareto Optimality

An important concept of multiobjective optimization is that of domination. Below, a formal definition of domination is given in the context of maximization problems [19]. The definition is easily extended to minimization problems.

A solution x_i is said to dominate x_j if:

$\forall\, k \in 1, 2, \ldots, M,$ $\quad f_k(x_i) \geq f_k(x_j)$ and

$\exists\, k \in 1, 2, \ldots, M$ such that $\quad f_k(x_i) > f_k(x_j)$.

where, M is number objective function. This concept can be explained using a two-objective optimization problem that has five different solutions, as shown in Figure 2 (example taken from [19]). Let us assume that the objective function f_1 needs to be maximized while f_2 needs to be minimized.

Five solutions having different values of the objective functions are shown. Evidently, solution 1 dominates solution 2 since the former is better than the latter on both objectives. Again solution 5 dominates 1, but 5 and 3 do not dominate each other. Intuitively, we can say that, if a solution 'a' dominates another solution 'b', then the solution 'a' is better than 'b' in multiobjective optimization. Thus, the concept of domination allows us to compare different solutions with multiple objectives. It may be noted that the dominance relation is irreflexive, asymmetric, and transitive in nature.

Assume a set of solutions *P*. The solutions of *P* that are not dominated by any other solution in *P*, form the non-dominated set. The rank of a solution *x* in *P* is defined as the number of solutions in *P* that dominate *x*. In Figure 2, solutions 3 and 5 are in the non-dominated set, and their ranks are 0. The non-dominated set of the entire search space *S* is the globally Pareto-optimal set [19].

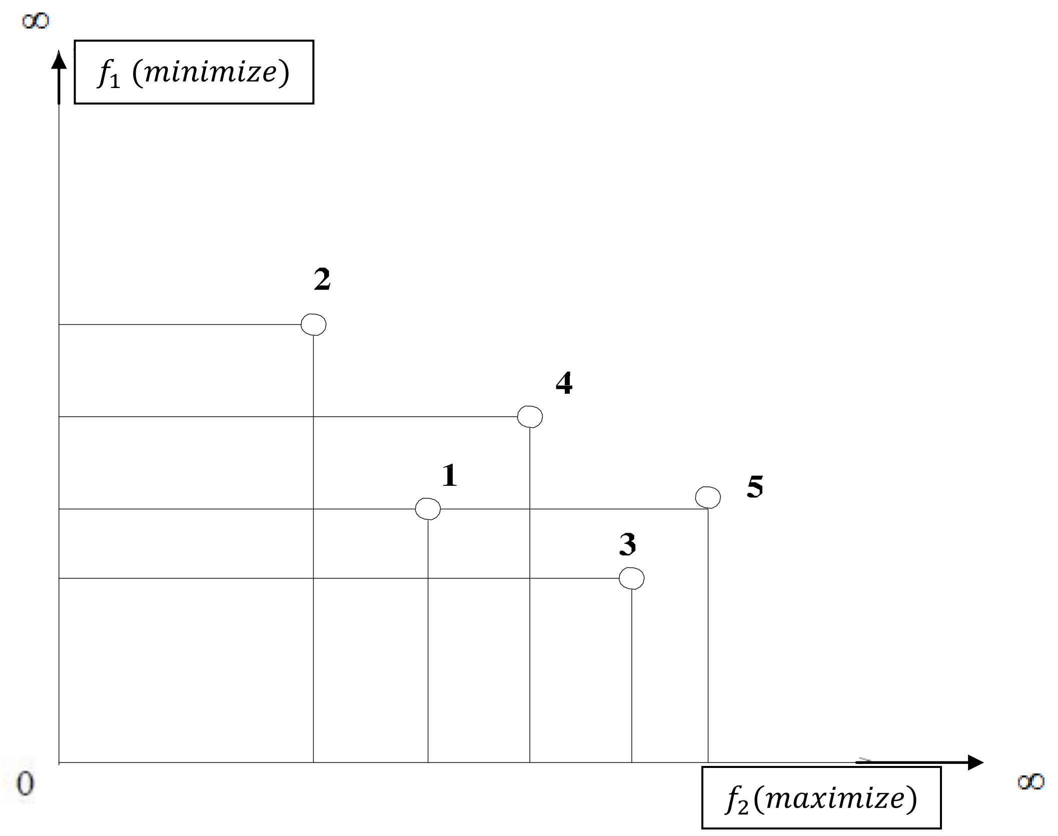


Figure 2: Map of a Two-objective Optimization Problem [19]

1.4 Non-dominated Sort

The population is sorted using the so called fast-non-dominated-sort [24]. For each individual *i*, an integer value holding the number of solutions that dominate *i* is created (domination count) and a set S_i with the individuals dominated by the individual *i* is calculated. With those parameters, each individual is assigned a rank representing the front to which it belongs. The Pareto front has rank 0. Those individuals dominated only by individuals from the Pareto front have rank 1. Generalizing, the individuals dominated only by individuals of rank *r* have rank *r+1*. The best solutions have always rank 0 with this approach [24].

1.5 Related Works On Bacteria Foraging Optimization

So far, several practical applications in different fields such as layout problems, network design problem, the blackboard wiring problem in electronics, the arrangement of electronic components in printed circuit boards and in microchips, machine scheduling in manufacturing, load balancing and task allocation in parallel and distributed computing, statistical data analysis, information retrieval, and transportation [3] have been introduced. Since, such problems are NP-hard, solutions can not be achieved in reasonable time, then there are some combinatorial optimization algorithms such as bacteria foraging optimization, genetic algorithm, particle swarm optimization, etc. to find the reasonable solutions in less time.

Bacteria Foraging Optimization Algorithm (BFOA) is a new bio-inspired optimization algorithm which has been developed to make a bridge between microbiology and engineering. The BFO algorithm mimics some characteristics of bacteria foraging such as chemo-taxis, metabolism, reproduction, and quorum sensing. The BFO has been introduced by Passino in 2002 [5], and it consists of four steps (mechanisms) namely: 1) 'chemo-tactic', 2) 'swarm', 3)'reproduction', 4)'elimination-dispersal' which is a new approach to solve complicated optimization problems. The detailed description will be given in chapter 2.

Jing Dang et al. [20] have proposed a paper about Bacterial Foraging Optimization (BFO) algorithm. This is a biologically inspired computation technique which is based on mimicking the foraging behavior of E.Coli bacteria. During the lifetime of E.coli bacteria, they undergo different stages such as chemotaxis, reproduction and elimination-dispersal. BFO algorithm was implemented various real world problems.

Kim suggested that the BFO could be applied to find solutions for difficult engineering design problems.

G.Naresh et al. [21] have proposed a novel approach based on BFO which has been successfully employed to solve Economic Load Dispatch (ELD) problem including valve point effects. The proposed algorithm has been tested for a test system with 3 and 13 generating units and the results thus obtained are compared with the results of earlier methods (PSO and GA) available in the literature. As compared to other two, the BFO is easy to implement and there are few parameters to adjust. Therefore, BFO has been successfully applied in many areas of power system. From the outcome of the results, it is shown that the proposed algorithm is very effective in giving quality solutions for ELD problems. Moreover, it also reveals that the fuel costs are reduced.

R. Vijay [22] has proposed optimal multi-objective design of robust multi-machine power system stabilizers (PSSs) using Bacterial Foraging Algorithm. In this paper, Eigenvalue analysis under different operating conditions reveals that undamped and lightly damped oscillation modes are shifted to a specific stable zone in the s-plane. These results show the potential of BFO algorithm for optimal design of PSS parameters. The nonlinear time-domain simulation results show that the proposed PSSs work effectively over a wide range of loading conditions and system configurations.

In the rest of this thesis, an introduction of bacteria foraging optimization algorithm and multi-objective bacteria foraging algorithm will be presented in section 2. In section 3, a new approach in order to solve QAP and mQAP by bacteria foraging algorithm will be proposed. In section 4, experimental results of proposed BFOA and multiobjective BFOA will be depicted. The results shows that proposed algorithm can solve QAP and mQAP, where the results are reasonable and compatible.

\

Chapter 2

BACTERIA FORAGING ALGORITHM

2.1 Bacteria Foraging Optimization Algorithm (BFOA)

The Bacterial Foraging Optimization Algorithm (BFOA) is one the nature-inspired optimization algorithms, which is inspired from bio mimicry of the E-coli bacteria. The BFOA is introduced by Kevin M. Passino in 2002 [5], and the main idea behind, is to eliminate those bacteria which have weak foraging methods and following up those bacteria which have breakthrough foraging methods to maximize energy obtained per unit time. In the execution of BFOA, each bacterium contacts with other bacteria by sending signal, in which bacteria move to the next step to collect nutrient if previous factors have been satisfied. The basis of BFOA contains four principle steps: 1) ‘chemo-tactic’, 2) ‘swarming’, 3) ’reproduction’, and 4) ‘elimination-dispersal’ [10].

2.1.1 Chemo-tactic

In biological point of view, this process is the movement of bacteria for gathering food. The E-coli bacterium is able to move in two diversity ways, swimming and tumbling, and it alternates between these two modes of operation. In the swimming way, the bacterium swims in the same direction to search for food, and in the tumbling way, it changes the direction to another direction. Assume $\theta^i(j,k,l)$ shows the current position in i^{th} bacterium, j^{th} chemo-tactic step, k^{th} reproduction step, and l^{th} elimination-dispersal event, the position of bacterium in the next chemo-tactic step by tumbling is as follows [10]:

$$\theta^{i}\,(j+1,k,l) = \mathrm{\theta^{i}(j,k,l)} + \mathrm{C(i)}\frac{\Delta\mathrm{(i)}}{\sqrt{\Delta^{\mathrm{T}}\mathrm{(i)}\Delta\mathrm{(i)}}} \tag{3}$$

where, $\mathrm{C(i)}$ shows the size of the step taken in the random direction specified by the tumble for i^{th} bacterium, $\Delta(i)$ indicates a vector in the random direction in population size whose elements lie in [-1, 1], and $\Delta^{T}(i)$ shows transposed randomize vector of direction $\Delta(i)$.

2.1.2 Swarm

In the section 2.1.1 discussion was for the case where no cell-released attractants are used to signal other cells that they should swarm together. Here, we will also have cell-to-cell signaling via an attractant and will represent that with $J_{cc}\left(\theta, \theta^{i}(j,k,l)\right)$, $i = 1, 2, \dots, S$, for the i^{th} bacterium [5,10]. Let

$$\mathrm{d_{attract}} = 0.1$$

be the depth of the attractant released by the cell and

$$\mathrm{w_{attract}} = 0.2$$

be a measure of the width of the attractant signal. The cell also repels a nearby cell in the sense that it consumes nearby nutrients and it is not physically possible to have two cells at the same location. To model this, let

$$\mathrm{h_{repelent}} = \mathrm{w_{repelent}}$$

be the height of the repellant effect. And

$$\mathrm{w_{repelent}} = 10$$

be a measure of the width of the repellant. The values for these parameters are simply chosen to illustrate general bacterial behaviors [5]. Let:

$$\begin{aligned} J_{cc}\big(\theta, P(j,k,l)\big) &= \sum_{i=1}^{S} J_{cc}^{i}\left(\theta, \theta^{i}(j,k,l)\right) \\ &= \sum_{i=1}^{S}\left[-d_{attract}\exp\left(-w_{attract}\sum_{m=1}^{P}\left(\theta_m - \theta_m^{i}\right)^2\right)\right] \\ &+ \sum_{i=1}^{S}\left[-h_{repelent}\exp\left(-w_{repelent}\sum_{m=1}^{P}\left(\theta_m - \theta_m^{i}\right)^2\right)\right] \qquad (4) \end{aligned}$$

denote the combined cell-to-cell attraction and repelling effects, where $\theta = [\theta_1, \dots, \theta_p]^T$ is a point on the optimization domain and θ_m^i is m^{th} component of the i^{th} bacterium position θ^i. Note that as each cell moves, so does its $J_{cc}^{i}\left(\theta, \theta^{i}(j,k,l)\right)$ function, and this represents that, it will release chemicals as it moves. Due to the movements of all the cells, the $J_{cc}\big(\theta, P(j,k,l)\big)$ function shows if many cells come close together there will be a high amount of attractant and hence an increasing likelihood that other cells will move toward the group [5].

2.1.3 Reproduction

In this part, those bacteria which have enough nutrient will be reproduced and others will be eliminated. The healthier bacteria also will be duplicated to the other half of the population which are less healthy, so that the population keep constant during process.

2.1.4 Elimination-dispersal

During the process, the population may eventually change their positions. In fact, when density of bacteria become high in a small area, then the temperature of that location will be increased. In this case, the process of algorithm may kill bacteria in high temperature in high density of bacteria location. As a result, it deals to apply elimination-dispersal event to relocate the bacteria in different environments. The elimination-dispersal also helps to take away from local optima.

The pseudo code related to BFOA is as follows [10]:

Parameters:

1. Parameters initialization $p, S, N_c, N_s, N_{re}, N_{ed}, P_{ed,}\ \theta^i, C(i)(i = 1, 2, \dots, S)$.

   where,

   p: shows dimension of search space.

   S: shows population of bacteria.

   N_c: shows chemo-tactic steps per bacterium lifetime.

   N_s: when the length of swim is going up, the amount of this parameter restricts it.

   N_{re}: shows the reproduction phases.

N_{ed}: shows the elimination-dispersal phases.

P_{ed}: shows a probability for eliminated-dispersed.

$C(i)(i = 1, 2, \ldots, S)$: shows tumbling phase size.

Algorithm:

2. Elimination dispersal counter: $l = l + 1$
3. Reproduction counter: $k = k + 1$
4. Chemo-tactic counter: $j = j + 1$

a) For $i = 1, 2, 3, \ldots, S$ take the chemo-tactic phase as follows.

b) Calculate the fitness function, $J(i, j, k, l)$.

Let, $J(i, j, k, l) = J(i, j, k, l) + J_{cc}(\theta^{i}(j, k, l), P(j, k, l))$ (add on the cell-to-cell attractant effect to the nutrient concentration).

c) Let $J_{last} = J(i, j, k, l)$ it will save the value since to find a better cost via a run.

d) Tumble: create the random vector $\Delta(i) \epsilon R^{p}$ with each element $\Delta_m(i), m = 1, 2, \ldots, p$ a random number on [-1, 1].

e) Move: using Equation (3)

This results in a step of size $C(i)$ in the direction of the tumble for bacterium i.

f) Compute $J(i, j + 1, k, l)$ and let

$$J(i, j + 1, k, l) = J(i, j, k, l) + J_{cc}(\theta^{i}(j + 1, k, l), P(j + 1, k, l)).$$

g) Swim

g (I). Let m=0.

g (II). While m<N_s(if did not decrease too long).

- Let m=m+1(counter for swim length).

- If $J(i, j+1, k, l) < J_{last}$, let $J_{last} = J(i, j+1, k, l)$ and by using Equation (3), $J(i, j+1, k, l)$ will be computed as phase [f].
- Else, let m=N_s.

h) Go to the next bacterium $i+1$ if $i+1 \neq S$ (i.e., go to [b]).

5. If $J < N_c$, go to phase (4).
6. Reproduction

   6(I). For the given k and l, and for each $i = 1, 2, 3, \dots, S$, let

$$J_{health}^{i} = \sum_{j=1}^{N_c} J(i, j, k, l)$$

   6(II). The S_r number of the bacteria will be died which have highest J_{health} values, and the rest of them will be splitted.

7. If $k < N_{re}$, go to the step 3.
8. Elimination-dispersal:

   For $i = 1, 2, 3, \dots, S$, delete and distribute each bacterium with probabilityP_{ed}.

   If $l < N_{ed}$, then go to the step 2; otherwise end.

Figure 7 and 8 shows flowcharts of BFO algorithm based on above pseudo code. In these figures, Figure 8 is the continued flowchart of Figure 7.

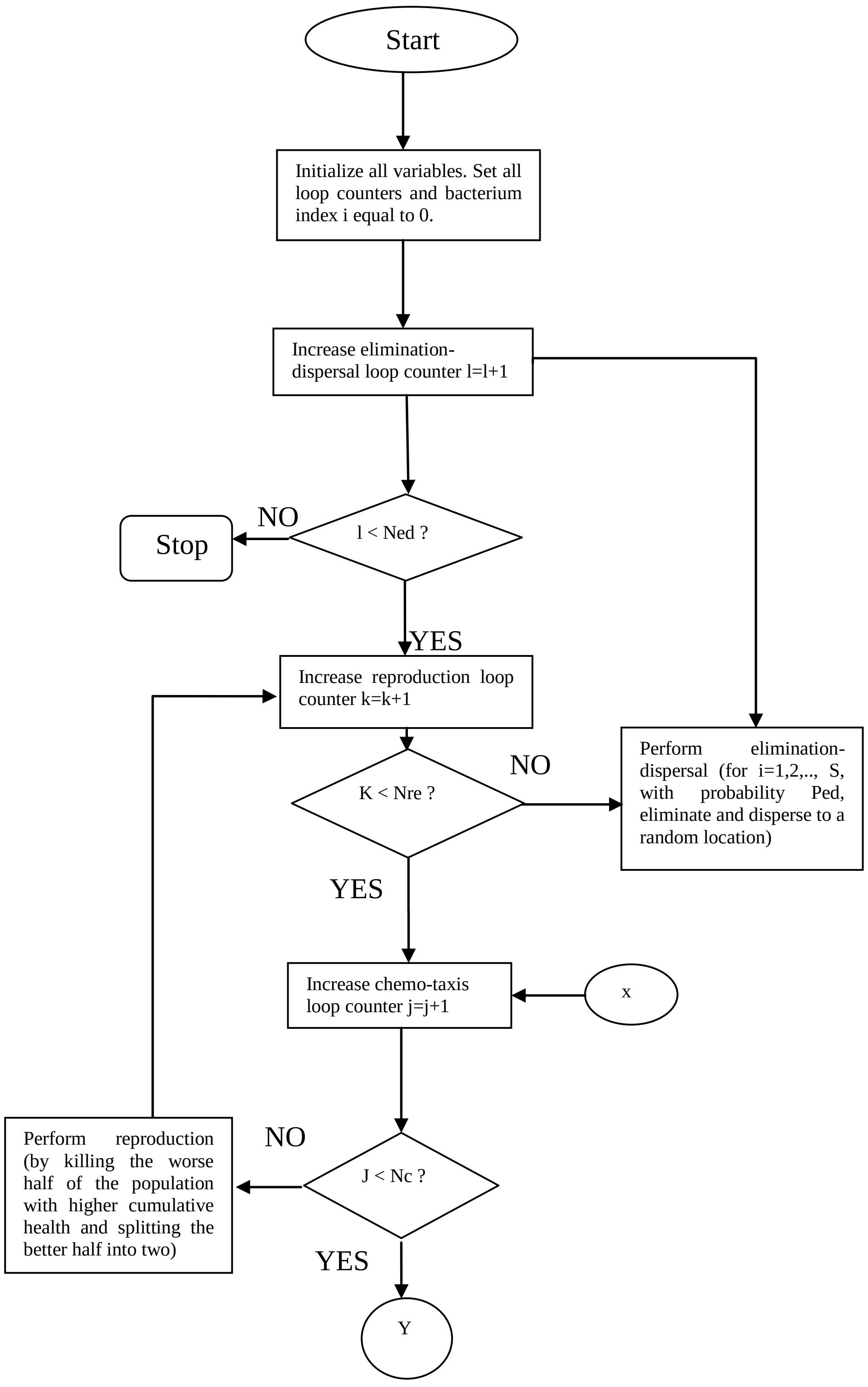
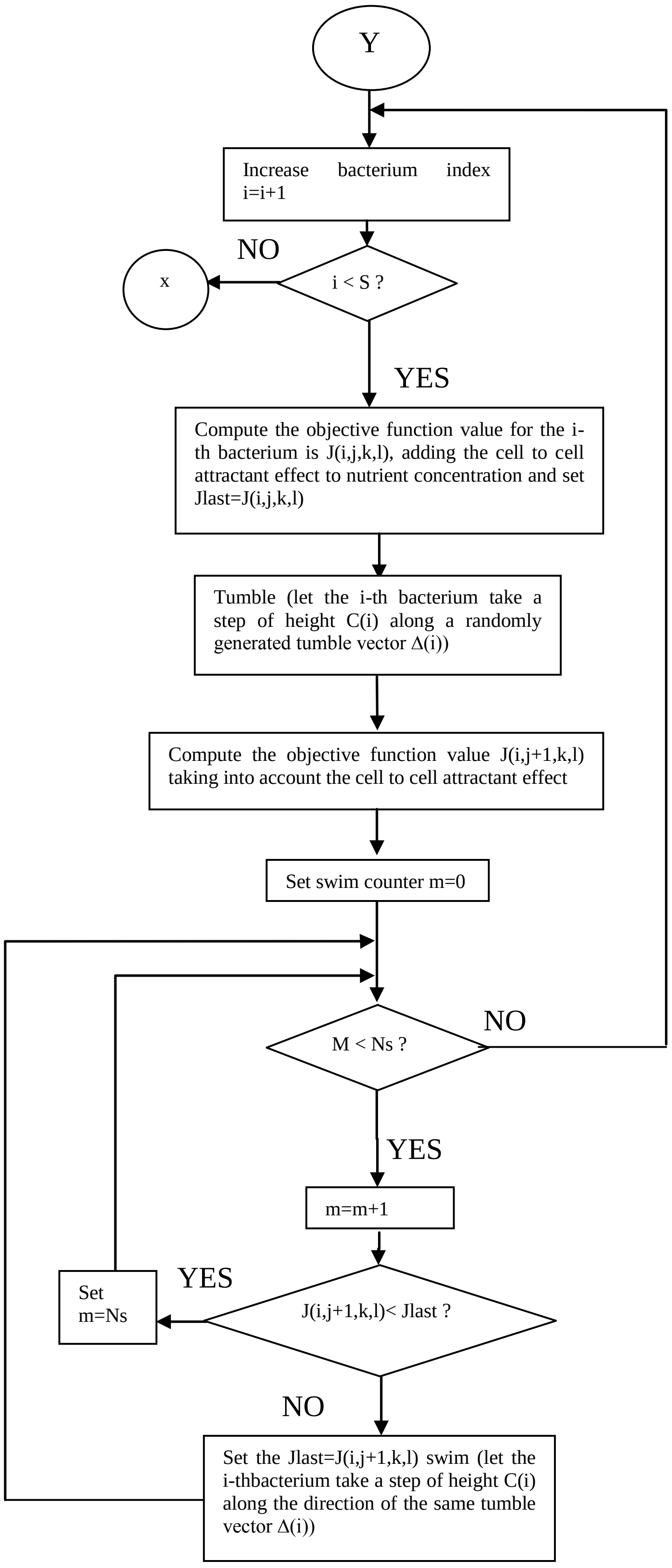

Start
Initialize all variables. Set all loop counters and bacterium index i equal to 0.
Increase elimination-dispersal loop counter l=l+1
l < Ned ?
NO
Stop
YES
Increase reproduction loop counter k=k+1
K < Nre ?
NO
Perform elimination-dispersal (for i=1,2,.., S, with probability Ped, eliminate and disperse to a random location)
YES
Increase chemo-taxis loop counter j=j+1
x
J < Nc ?
NO
Perform reproduction (by killing the worse half of the population with higher cumulative health and splitting the better half into two)
YES
Y

Figure 3: Flowchart of Bacteria Foraging Algorithm (a)



Y
Increase bacterium index i=i+1
NO
x
i < S ?
YES
Compute the objective function value for the i-th bacterium is J(i,j,k,l), adding the cell to cell attractant effect to nutrient concentration and set Jlast=J(i,j,k,l)
Tumble (let the i-th bacterium take a step of height C(i) along a randomly generated tumble vector Δ(i))
Compute the objective function value J(i,j+1,k,l) taking into account the cell to cell attractant effect
Set swim counter m=0
NO
M < Ns ?
YES
m=m+1
YES
Set m=Ns
J(i,j+1,k,l)< Jlast ?
NO
Set the Jlast=J(i,j+1,k,l) swim (let the i-thbacterium take a step of height C(i) along the direction of the same tumble vector Δ(i))

Figure 4: Flowchart of Bacteria Foraging Algorithm (b) [10]

2.2 Multiobjective Bacteria Foraging Optimization (MOBFO)

In the BFO algorithm bacteria attempt to find vast number of nutrient substance and avoid from noxious substrates. In this case, there is just one objective which explores the search process. Instead Multi-objective Bacterial Foraging Optimization (MOBFO) algorithm is inspired for solution of multiobjective optimization problems. The main aim of multiobjective optimization problem is to find all values which are possibly satisfied to all fitness functions. Since different decision makers have different ideas about fitness functions, it is not easy to choose a single solution for multiobjective optimization problems without interaction with the decision makers. Thus, all it could do is to show the set of Pareto-optimal solutions to decision makers. The main target of multiobjective optimization problems is to obtain a non-dominated front which is close to the true Pareto front (Section 1.3). Thereafter, the MOBFOA with integration between health sorting approach and Pareto dominance mechanism to solve multi-objective problems is proposed [11, 12].

The pseudo code related to MOBFOA [11]:

1. Parameters initialization: $p, S, N_c, N_{ed}, P_{ed,}P_g, M, \theta^i, C(i)(i = 1, 2, \dots, S)$, set rank for all bacteria to 1.

   where,

   p: is defined as dimension of search space,

   S: is defined population of bacteria.

   N_c: is defined as chemo-tactic phases for each bacterium lifetime

   N_{ed}: is defined as elimination-dispersal phases,

P_{ed}: is probability for eliminated-dispersed.

P_g: guide probability,

M: number of fitness functions,

$C(i)(i = 1, 2, \dots, S)$: is defined as tumbling phase size.

2. Elimination-dispersal counter: $ell = ell + 1$
3. Chemo-tactic counter: $j = j + 1$
4. Take the chemo-tactic phase for i^{th} bacterium, $i = 1, 2, 3, \dots, S$ as follows.
5. Compute the fitness function $J_b(i, j, k)$, in which b is fitness function, $b = 1, 2, \dots, M$.
6. Tumble: generate a random number p.

   If $p < P_g$, and rank of its bacterium is greater than 1, generate $\Delta(i)$, which is a unit vector towards another bacterium belonging to a front whose rank is less (means quality is better). The index of the new bacterium is chosen randomly. Suppose r^{th} bacterium is chosen at random and it belongs to a front whose rank is less than that of i^{th} bacterium. Then,

   $\Delta(i) = \theta(r, j, ell, b) - \theta(i, j, ell, b)$.

   Else,

   Generate a random vector $\Delta(i) = R^p$ with each element $\Delta_m(i), m = 1, 2, \dots, p$, which random number lie [-1, 1].
7. Move:

$$\theta^i\,(j+1, k, l) = \theta^i(j, k, l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$$

8. Go to the next bacterium $(i + 1)$ if $i \neq S$ (i.e. go to (b) to process the next bacterium).

9. Store all these results with old ones in the memory, these results will be sorted on the basis of non-dominated sorting.
10. Those which have better rank will continue their life for the next iteration on $j + 1$.
11. If j$< N_c$ go to step 3, and start next chemo-tactic steps till number of reproduction steps are reached.
12. Elimination-dispersal: For $i = 1, 2, 3, \dots, S$ with probabilityP_{ed}, If $ell < N_{ed}$, start from phase 2, otherwise end.

Chapter 3

PROPOSED SINGLE AND MULTIOBJECTIVE OPTIMIZATION ALGORITHMS FOR THE SOLUTION OF QAP AND mQAP

3.1 Introduction

Bacteria foraging algorithm is one of the bio inspired algorithms which can be solved QAP. Since QAP is one of the nonlinear problems, most probably reasonable solutions can be achieved with deterministic algorithms [14].

In this thesis, BFO and MOBFO have been used for the solution of QAP and mQAP, respectively. In the updating part of the BFO different operators of GA [23] have been used such as crossover and mutation. Also, for improving the solutions with a local search, tabu search [18] have been applied. As long as there are two concepts to optimize (single objective and multi-objective), first of all the definition of each will be presented below and then the corresponding algorithm for the solution of problems in different concepts will be explained.

3.2 Proposed BFO Algorithm for the Solution of QAP

In the case of single objective QAP the aim is to find one compatible solution in which the cost between facilities and locations becomes as minimum as possible. In fact, by the modified BFO and using GA techniques such as crossover and mutation, as well as Tabu search algorithm in this algorithm, after several iterations a compatible solution will be obtained. Basically, the modified BFO algorithm has three main steps: 1) Chemo-tactic, 2) Reproduction, and 3) Elimination-dispersal (Figure 3.1).

The pseudo code related to modified BFO is as follows:

```
1. Initialization parameters $p, S, N_c, N_{re}, N_{ed}, S_r, P_{ed}$.
   where,
   $p$: is defined as dimension of search space.
   S: is defined as population bacteria.
   $N_c$: is defined as chemo-tactic phases.
   $N_{re}$: is defined as reproduction phases.
   $N_{ed}$: is defined as elimination-dispersal phases.
   $S_r$= $S/2$: is defined as bacteria split.
   $P_{ed}$: is defined as eliminated-dispersed probability.
2. Make a random permutation for $i^{th}$ bacterium, $i = 1, 2, 3, \dots, S$, and compute the fitness function $J(i, j, k)$.
3. Get the minimum cost which obtained by fitness function and set it as best so far.
4. Elimination-dispersal counter: $ell = ell + 1$.
5. Reproduction counter: $k = k + 1$.
6. Chemo-tactic counter: $j = j + 1$.
   a) Take the chemo-tactic step for $i^{th}$ bacterium, $i = 1, 2, 3, \dots, S$.
   b) Apply crossover to every two bacteria or apply mutation for each bacterium $i^{th}$.
```

c) Compute the objective function $J(i, j, k)$.

d) Get minimum cost, if it is better than previous one, then replace it as best so far.

e) Go to next bacterium $(i + 1)$ if $i \neq S$ (i.e. go to (b) to process the next bacterium).

f) Get the minimum cost so far.

g) If j$< N_c$ go to the step 6, and start next chemo-tactic steps till number of chemo-tactic steps are reached.

7. Eliminate half of the bacteria and copy other half to this part which lead to population became stable in same number. If $k < N_{re}$ go to the phase 5,
8. Elimination-dispersal: For $i = 1, 2, 3, \dots, S$ with probability P_{ed}, delete and distribute the bacteria.
9. Get minimum best so far.
10. Apply robust Tabu search, if $ell < N_{ed}$, then go to the phase 4.
11. End.

Figure 9 shows flowchart of proposed BFO algorithm based on above pseudo code.

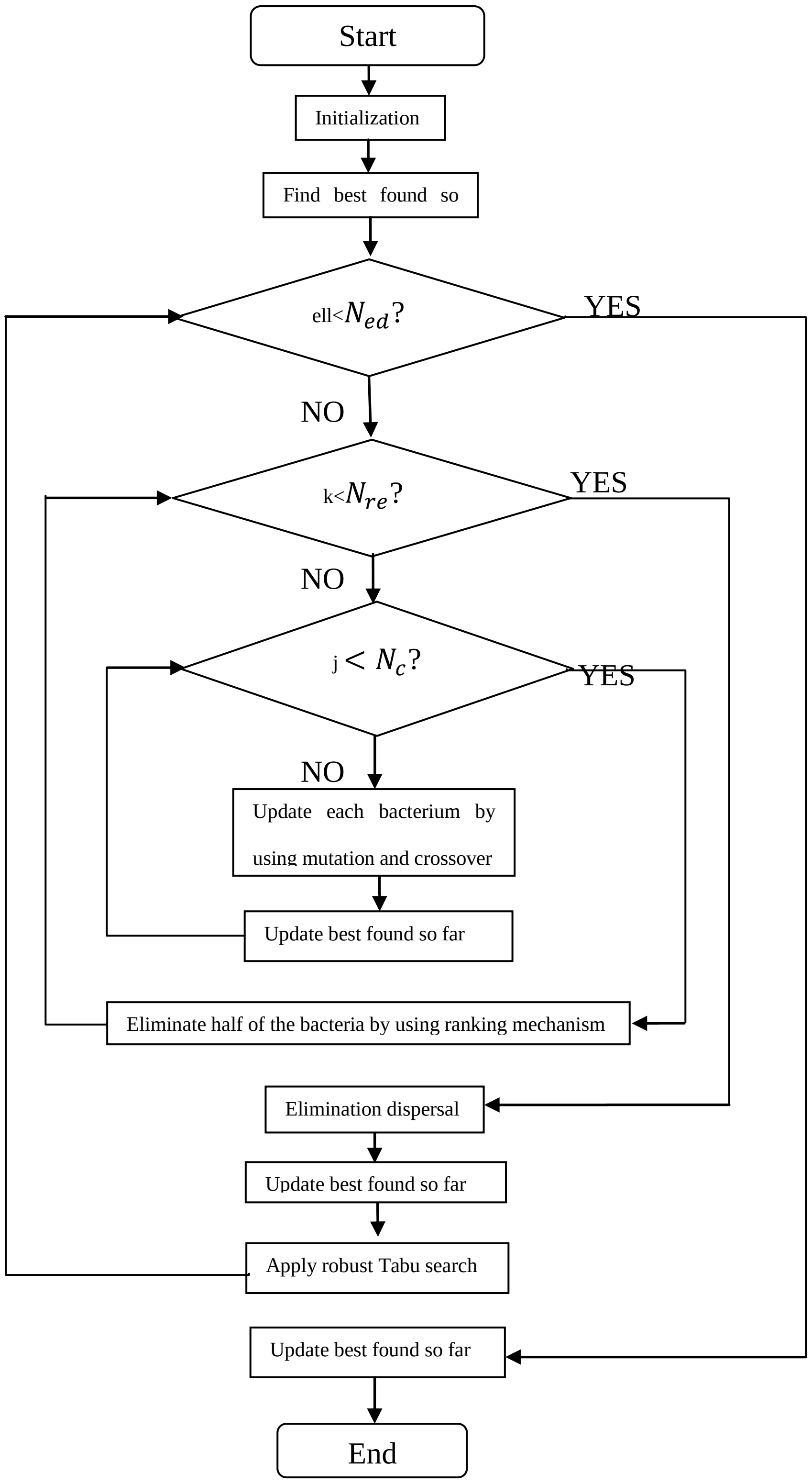


Figure 5: Flowchart of Proposed BFO Algorithm

3.2.1 Initialization

In the initialization part, the random permutation in *P* size for each bacterium will be generated. Then, the single objective QAP with one flow matrix and one distance matrix will be given. Next, the assignment cost of each facility to the corresponding location by using Equation (1) will be computed. This will continue until all the bacteria in the population are being computed. At the end of this step, the minimum cost through all population will be stored as best found so far.

3.2.2 Chemo-tactic

In this step, inversion mutation [24] and swap mutation [25] have been applied on permutations. Afterwards, the new population of bacteria (permutations) will be generated, and the new assignment cost will be computed for each bacterium. Additionally, the minimum assignment cost of new generation will be compared by the best found so far. If the new cost is less than that, the new cost will be replaced as best found so far. This step will be repeated until the end of the chemo-tactic loop.

The definition regarding to inversion mutation and swap mutation are as follows:

3.2.2.2 Mutation

The aim of mutation is making some modification on current permutation. There are several techniques for mutation. Here two methods of mutations have been defined as follows:

Swap mutation is one of the simplest mutation methods such that two locations of the chromosome will be selected and exchanged (Figure 5).

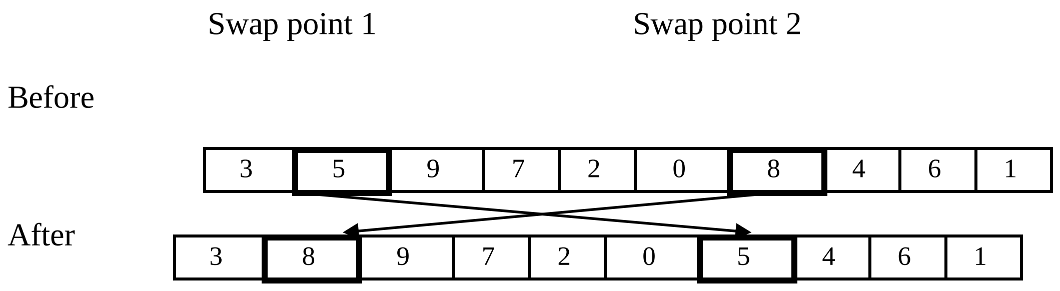


Figure 6: Swap Mutation on Random Permutation

In p/3 mutation [26], given a permutation size (*p*) is being divided by three, and the swap mutation will be applied *p/3* times (Figure 5).

In inversion mutation technique, two random points will be generated namely cut point 1 and cut point 2. After that, elements of this range will be reversed and replaced into offspring. Figure 6 depicts a simple example of this method:

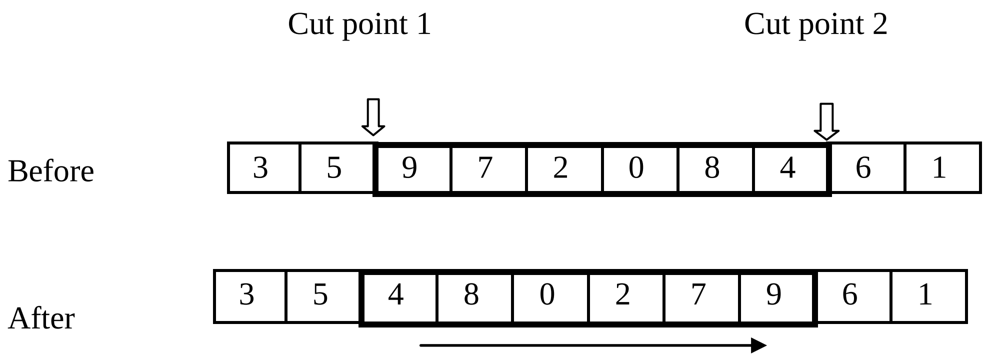


Figure 7: Inversion Mutation on Random Permutation

3.2.3 Reproduction

In this step, all the costs, which had been obtained in the previous step, will be sorted in ascending order and the first half of the population will be copied to the second half. Additionally, the second half will be eliminated from population. Note, for more convenience the number of bacteria (population) have been set to an even number such that in duplicating time both parts are same, and the population will be constant. This step goes until the end of the reproduction iteration.

3.2.4 Elimination-dispersal

In this part, P_{ed} has been set to *0.25*. Basically, in each iteration a random number will be generated which had lied on *(0, 1).* Afterwards, a simple comparison implies such that if the random number is smaller than P_{ed}, then initialization part will be repeated here, otherwise the algorithm goes on with old population. Essentially, this step will be avoided of local optima.

3.2.5 Robust Tabu Search

Local search algorithms are widely applied to numerous hard computational problems. Examples of local search algorithms are WalkSAT and the tabu search (TS) algorithm for the Traveling Salesman Problem (TSP). Since, QAP is defined as TSP, so essentially, TS can work with QAP. Consequently, the best found so far permutation will be given to Tabu search algorithm, and obtained result (permutation) by Tabu search will be replaced instead of worse permutation, regarding to its cost, in the population. This will be continued until the end of the elimination dispersal loop.

Tabu search is a metaheuristic local search algorithm that can be used for solving combinatorial optimization problems. Tabu search uses a local or neighborhood search procedure to iteratively move from one potential solution x to an improved solution x' in the neighborhood of x, until some stopping criterion has been satisfied (generally, an attempt limit or a score threshold). Tabu search carefully explores the neighborhood of each solution as the search progresses. The solutions admitted to the new neighborhood, $N^*(x)$, are determined through the use of memory structures. Using these memory structures, the search progresses by iteratively moving from the current solution x to an improved solution x' in $N^*(x)$ [14, 18].

The memory structures used in tabu search can be divided into three categories:

- Short-term: The list of solutions recently considered. If a potential solution appears on this list, it cannot be revisited until it reaches an expiration point.
- Intermediate-term: A list of rules intended to bias the search towards promising areas of the search space.
- Long-term: Rules that promote diversity in the search process.

Pseudo code of tabu search algorithm for minimizing problems has been defined as follows [17]:

Algorithm:

```
1.  Input: Tabu $\text{list}_{\text{size}}$
2.  Output: best solution $S_{\text{best}}$
3.  $S_{\text{best}} =$ Construct initial solution ()
4.  tabu list = 0
5.  While (Not Stop condition)
6.      Candidate list = 0
7.      For $(S_{\text{condidate}} \in \text{SBest}_{\text{neighborhood}})$
8.          If (Not contains any features ($S_{\text{condidate}}$, tabu list))
9.            Candidate list = $S_{\text{condidate}}+S_{\text{condidate}}$
10.         End
```

```
11.      End
12.     $S_{condidate}$ = local best candidate (candidate list)
13.      If (cost ($S_{condidate}$) ≤ cost ($S_{best}$))
14.        tabu list = feature differences ($S_{condidate}$, $S_{best}$)
15.          $S_{best}$ = $S_{condidate}$
16.                  While (tabu list > tabu $list_{size}$)
17.                Delete feature (tabu list)
18.            End
19.       End
20. End
21. Return $S_{best}$
```

Lines 1-4 represent some initial setup, respectively creating an initial solution (possibly chosen at random), setting that initial solution as the best seen to date, and initializing an empty tabu list. In this example, the tabu list is simply a short term memory structure that will contain a record of the elements of the states visited.

The proper algorithm starts in line 5. This loop will continue searching for an optimal solution until a user-specified stopping condition is met. In line 6, an empty candidate list is initialized. The neighboring solutions are checked for tabu elements in line 8. If the solution does not contain elements on the tabu list, it is added to the candidate list (line 9).

The best candidate on the candidate list is chosen in line 12 (generally, solutions are evaluated according to a provided mathematical function, which returns a fitness score). If that candidate has a higher fitness value than the current best (line 13), its features are added to the tabu list (line 14) and it is set as the new best (line 15). At this point, if the tabu list is full (line 16), some elements will be allowed to expire (line 17). Generally, elements expire from the list in the same order they are added.

This process continues until the user specified stopping criterion is met, at which point, the best solution seen during the search process is returned (line 21).

3.3 Proposed MOBFO Algorithm for the Solution of mQAP

In the case of multiobjectives QAP (mQAP) the aim is to find a set of non-dominated solutions in which the cost between facilities and locations became minimized. In fact, the modified BFO by using *p/3* mutation, inversion mutation, swap mutation, and ULX and to improve the non-dominated set, tabu search algorithm in elimination-dispersal part, will achieve good solutions after several iterations. Basically, the modified multiobjectives BFO (MOBFO) algorithm has three main steps as BFO: 1) Chemo-tactic, 2) Reproduction, and 3. Elimination-dispersal (Figure 10).

The pseudo code related to MOBFO is as follows:

1. Initialize parameters $p, S, N_c, N_{re}, N_{ed}, S_r, P_{ed}, M,$ set rank for all bacteria to 1.

   where,

   p: is defined as dimension of search space,

   S: is defined as population of bacteria.

   N_c: is defined as chemo-tactic phases.

   N_{re}: is defined as reproduction phases.

   N_{ed}: is defined as elimination-dispersal phases.

S_r= *S/2*: is defined as bacteria split.

P_{ed}: is defined as eliminated-dispersed probability.

M: is number of fitness functions.

2. Make a random permutation for i^{th} bacterium, $i = 1, 2, 3, \dots, S$, and compute the fitness functions $J_b(i, j, k)$, in which b is objective functions, $b = 1, 2, \dots, M$.
3. Get the non-dominated set which obtained by fitness functions and set it as non-dominated set.
4. Elimination-dispersal counter: ell= $ell + 1$
5. Reproduction counter: $k = k + 1$
6. Chemo-tactic counter: $j = j + 1$
    a) Take the chemo-tactic step for i^{th} bacterium, $i = 1, 2, 3, \dots, S$ as follows.
    b) Apply crossover and mutation
    c) Calculate the fitness functions $J_b(i, j, k)$,
    d) Get non-dominated set,
    e) Go to next bacterium $(i + 1)$ if $i \neq S$ (i.e. go to (b) to process the next bacterium),
    f) Store all these results with old ones in the memory, these results will be sorted as basis non-dominated sorting,
    g) Those which have better rank will continue their life for the next iteration on $j + 1$, if j< N_c go to the step 6,
7. Eliminate half of the bacteria and copy other half to this part which leads to population to become stable with the same number. If $k < N_{re}$ go to the step 5,
8. Elimination-dispersal: For $i = 1, 2, 3, \dots, S$ with probabilityP_{ed}, delete and distribute the bacteria.
9. Get non-dominated set.

```
10. Get best permutation and apply robust tabu search. If $ell < N_{ed}$, then go to step
    4.
11. Update non-dominated set.
12. End.
```

Figure 10 shows flowchart of proposed MOBFO algorithm based on above pseudo code.

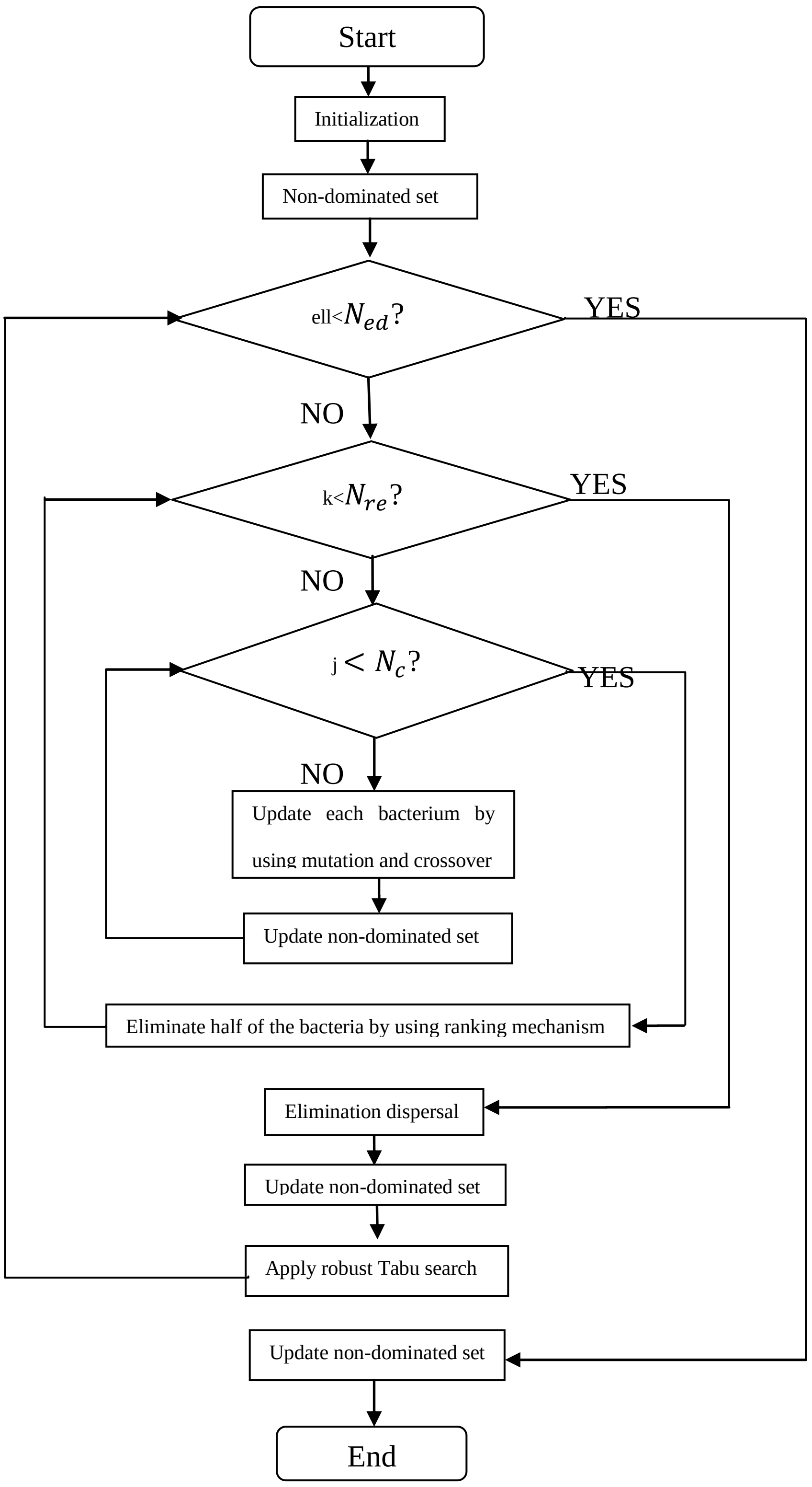


Figure 8: Flowchart of Proposed MOBFO Algorithm

3.3.1 Definition

In the MOBFO algorithm, additional update mechanisms such as *p/3* mutation, inversion mutation, swap mutation, and ULX [17] have been applied. In order to apply ULX, every two bacteria have been selected, and new population based on ULX will be generated. The definition of ULX is given in next section.

Since, in mQAP, the aim is to find a set of non-dominated solutions, the ranking mechanism has been used in the reproduction part. In this respect, first of all the population of bacteria with rank 0 will be sorted and then rank 1, and so on. Afterwards, the second half of population of the bacteria will be eliminated and first part will be copied to the second part.

3.3.2 Crossover

Crossover is one of the main operators of genetic search. In this operator a different solution (child) by exchanging some elements in two parents will be generated. Specifically, crossover is a random operator in which with inspire of both parents generates new features. So far, many methods of crossover have been introduced. Here are two methods of crossover which are used in the proposed algorithm.

Uniform Like crossover operator had introduced by Tate and Smith in 1995 [17], and mostly applies permutation based solution which is called uniform like crossover (ULX). Uniform crossover allows some flexibility, and different variations of the basic procedure are possible. The ULX works as follows:

First of all, those elements which are same in both parents (Figure 3) will be chosen and will be copied to child (offspring). After that, one of the parents will be chosen and the first element of it will be copied to the first element of child (it goes forward from left to right). Specifically, after moving one of the elements, the pointer jumps to another parent and selects next element. But, it should be taken to account that previous elements must not be the same, if so, then pointer will change the position to another parent. If again encountered with repetition then a new number in the permutation range will be generated, randomly. This will be continued till all the locations are being filled.

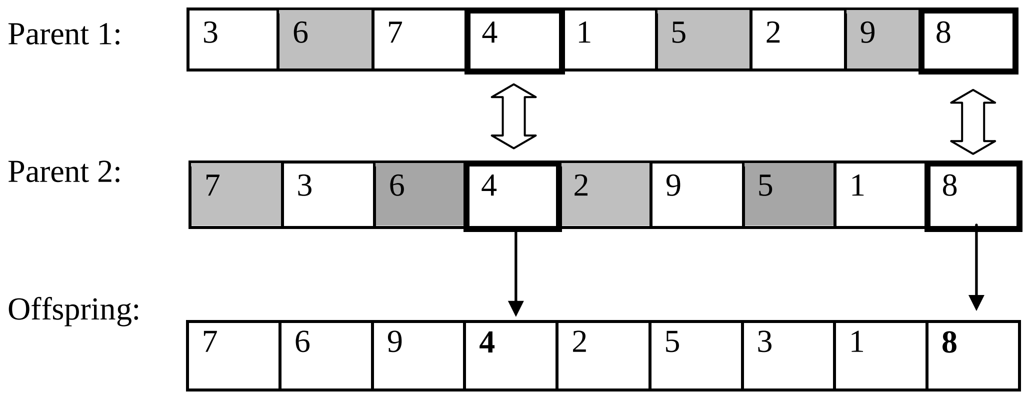


Figure 9: The Uniform Like Crossover with Random Permutation

Block crossover operator is achieved by some modification in the previous crossover operator (ULX) which is called randomized uniform like crossover (RULX) or block crossover (BX). The only difference between ULX and BX, is how to scan the position. Based on the previous description, in the ULX the order of scanning is fixed from left to right. On the other hand, in the BX the consideration of the position is done completely as a random process. Actually, the main aim in BX is adding more randomness and diversity to the offspring. The BX crossover works as follows:

Initially, parents will be divided to some elements or segments instead of single element which is called block. The block size is in the range of [*1*, *n/2*] (Figure 4). Clearly, if the number of elements become even, then the block size would be 2, otherwise one of the block size will be 3. Initially, one of the parents will be selected randomly, and the block will be copied to the offspring. After that, pointer switches to another parent and copies the second block. Similarly, if there existed any repetition element in the prior elements of offspring, it should switches to another parent and copies only one element from opposite block, otherwise will be continued forward. Yet, it might not be solved by switching the block for avoiding repetition. In this case, the new random number in the range of permutation size will be generated and copied to the offspring location. Finally, this process goes till all the blocks are being copied to the offspring.

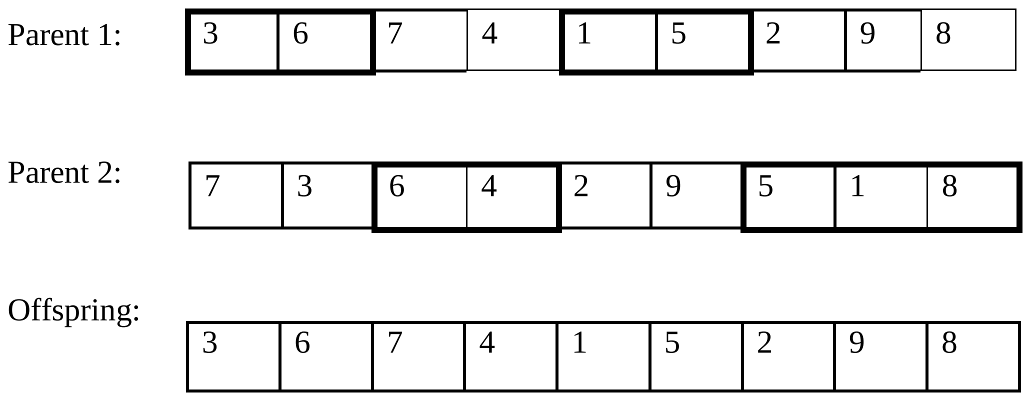


Figure 10: Block Crossover with Random Permutation

Chapter 4

EXPERIMENTAL RESULTS

Problems from the well-known QAPLIB [16] are used here to evaluate the performance of the algorithms. The following parameter settings were used for each algorithm tested: $S = 100, N_c = 100, N_{re} = 4,\ N_{ed} = 10, P_{ed} = 0.25$. Basically, different methods and techniques have been used to show how they affect the performance of results. In the Table 1 the results of single objective optimization have been shown. In this table, the first column shows the problems, the second column shows the best results which have been obtained by using proposed algorithm, the third column shows the worst results, and the fourth column shows the optimal solutions for corresponding problems that have been obtained so far. The given results are collected by running each algorithm 10 times.

In Table 1, the standard deviation shows the difference between the results of 10 runs. The low standard deviation means the gathered solutions are very close to each other, and the high standard deviation means the gathered solutions are very far from each other. For example, in the problem Scr20 the standard deviation shows that the optimal results are very far from each other. The reason of high standard deviation is because of the hardness of the problem, or in other words, it is because of the fact that the distances between facilities and locations are very large.

Table 1: Result of Proposed Single Objective BFO

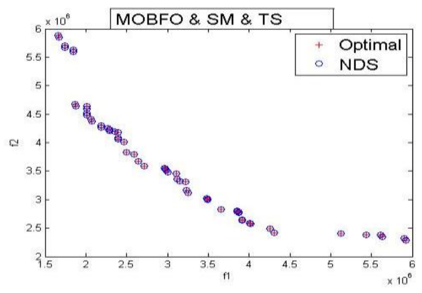
Problems	Swap mutation& Shift mutation & Robust Tabu search			Optimal
	Best	Worst	Standard deviation	
bur26f.dat	3782068	3835254	25192.3966865	3782044
esc16a.dat	68	68	0.0	68
esc16h.dat	996	996	0.0	996
esc32e.dat	2	2	0.0	2
esc32f.dat	2	2	0.0	2
esc128.dat	64	64	0.0	64
had12.dat	1652	1676	7.2	1652
had14.dat	2724	2724	0.0	2724
had16.dat	3720	3720	0.0	3720
had20.dat	6922	6992	21.0	6922
lipa20a.dat	3683	3683	0.0	3683
lipa20b.dat	27076	27076	0.0	27076
lipa50b.dat	1210244	1210244	0.0	1210244
scr12.dat	31410	32236	247.8	31410
scr20.dat	110030	139124	8572.74425141	110030

The results for 13 multiobjective QAP problems have been given by using the proposed MOBFOA. One of the multiobjective plots has been shown below, and the rest of them are in the Appendix.

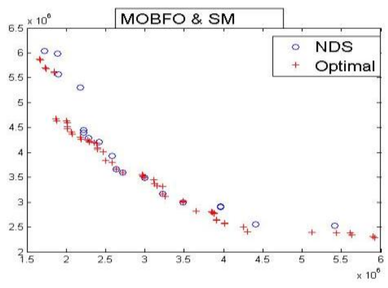
Figure 11 (a) shows optimal and non-dominated solution (NDS) found by using swap mutation with tabu search, and Figure 11 (b) performance of proposed algorithm without using tabu search depicted. Part (c) shows the performance of proposed algorithm by using *p/3* swap mutation with tabu search, and part (d) shows the results with only *p/3* swap mutation. For the part (e) the performance of proposed algorithm by using inversion mutation with tabu search have been given, and part (f) shows the results with only inversion mutation. Finally, part (g) shows performance of proposed algorithm by using ULX with tabu search together, and part (h) without using tabu search have been depicted.

Results with blue color shows achievements of the proposed algorithm (NDS), and the optimal results according to the QAPLIB with red color are given. Moreover, in the case of three objective plots which are depicted in last four plots, only the non-dominated set solutions have been shown.

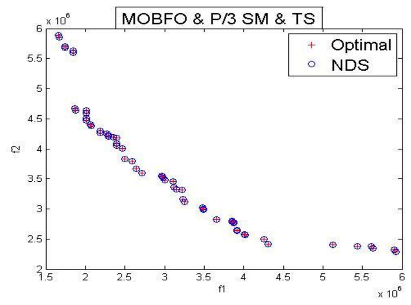
The gathered results by using proposed algorithm with swap mutation, *p/3* swap mutation, inversion mutation, and ULX, have been shown in the Figures. According to Pareto front shape which is appear in the figures, when the tabu search have been applied with each of the genetic operators, the Pareto front of NDS is very close to the Pareto front of optimal solutions (Figure 11 (a), (c), (e), (g)). On the other hand, the Pareto front shape which have been achieved by only using the swap mutation, *p/3* swap mutation, inversion mutation, and ULX, is not good as optimal solutions Pareto front (Figure 11 (b), (d), (f), (h)). The *p/3* operator that applied to the mQAP problems shows the best performance than all the others operators. For example, in problem KC10-2fl-1rl, the performance of NDS Pareto front in part (c) is almost same as optimal Pareto front (the error ratio and hyper volume in table 2 verifies this claim). Nevertheless, the performance of NDS Pareto front in part (h) shows gathered solutions by using ULX without tabu search is the worst one.



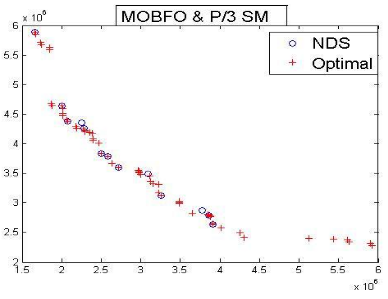
(a)



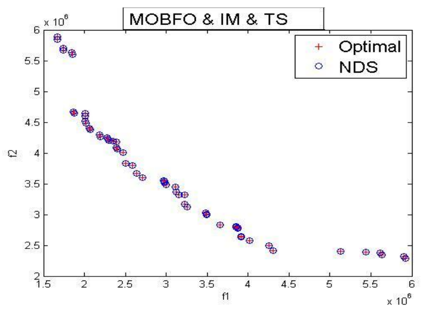
(b)



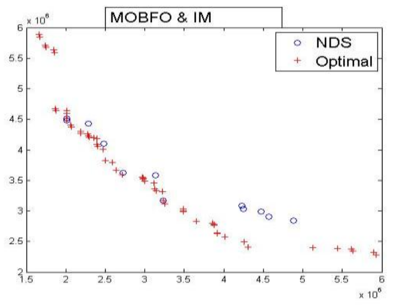
(c)



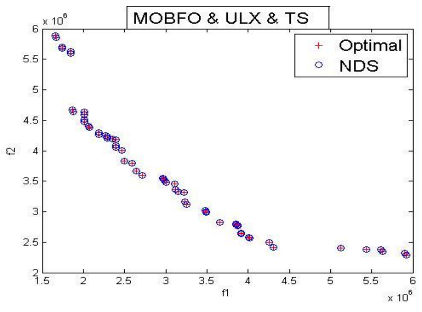
(d)



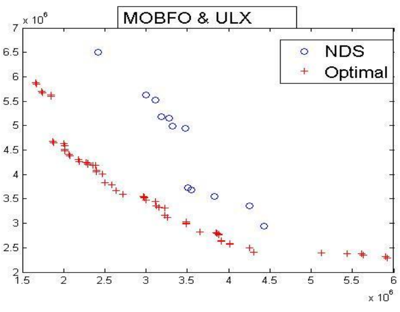
(e)



(f)



(g)



(h)

Figure 11: Problem KC10-2fl-1rl with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX

Table 2: ER and HV for Figure 11

KC10-2fl-1rl	Error Ratio	Hyper volume
Swap mutation	0	0.34025
Shift mutation	0	0.34295
p/3 swap	0	0.33900
ULX	0	0.33460

In Table 2, quality indicators error ratio and hyper volume have been given for problem KC10-2fl-1rl. The error ratio and hyper volume values indicate that how different the optimal and NDS solutions from each other. Specifically, the error ratio shows that NDS is exactly the same as the optimal ones, and the hyper volume value shows that the area covered by NDS is bigger than optimal ones. Furthermore, the low value of hyper volume means less difference between NDS and optimal Pareto fronts.

Chapter 5

CONCLUSION

Since, QAP is a NP-hard problem, solution to it can not be achieved in reasonable time, and a combinatorial optimization algorithm should be used to manage such obstacle. The bacteria foraging algorithm is one of the well-known combinatorial optimization algorithms which can find compatible solutions in reasonable time. In this thesis, a novel algorithm has been proposed for the solution of QAP and mQAP. Basically, this algorithm is based on bacteria foraging optimization which have been developed in two extents: single objective optimization and multiobjective optimization. For this purpose, Genetic Algorithm updating mechanisms such as uniform like crossover and shift mutation, and a kind of local search method as robust tabu search have been used to improve the solutions. Moreover, in the single objective optimization, the proposed algorithm attempts to find best solution. On the other hand, multiobjective optimization, algorithm tries to find a set of non-dominated solutions in reasonable time. Therefore, in the multiobjective concern, other techniques such as dominance have been applied to find a set of non-dominated solutions.

The proposed algorithm results showed that BFOA and MBFOA can give good results for the QAP and mQAP problems. Also, results showed that different GA operators help BFOA and MOBFOA to update the solutions and move towards the best ones. Specifically, in Table 1 results of single objective problems by using

BFOA have been shown that are exactly same as optimal solutions. The performance of Pareto fronts that have been achieved for the mQAP problems (Figure 11, 12, …) by using MOBFO showed that proposed method can give good results.

REFRENCES

[1] T. C. Koopmans and M. J. Beckmann, (1957). Assignment problems and the location of economic activities, Econometrica 25, 53–76.

[2] J. Knowles and D. Corne. (2002). Towards Landscape Analyses To Inform The Design of A Hybrid Local Search for The Multiobjective Quadratic Assignment Problem. Soft Computing Systems: Design, Management and Applications, pp. 271-279. IOS Press, Amsterdam.

[3] M. Zhao, A. Abraham, C. Grosan and H. Liu. (2008). A Fuzzy Particle Swarm Approach to Multiobjective Quadratic Assignment Problems: School of Software, Dalian University of Technology, 116620 Dalian, China., DOI 10.1109/AMS.169

[4] Rainer E. Burkard, Eranda C¸ ela Panos M. Pardalos. (1998)The Quadratic Assignment Problem, Spezialforschungsbereich F 003 “Optimierung und Kontrolle”, Projektbereich Diskrete Optimierung.

[5] K. M. Passino. (2002). Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Syst. Mag.*, vol. 22, no. 3, pp.52– 67.

[10] S. Das, A. Biswas, S. Dasgupta, and A. Abraham. (2009). Bacterial Foraging optimization Algorithm: Theoretical Foundations, Analysis, and

Applications,Dept. of Electronics and Telecommunication Engg, Jadavpur University, Kolkata, India.

[11] B.K. Panigrahi, V. R. Pandi, R. Sharma, Das, S. Das (2011). Multiobjective bacteria foraging algorithm for electrical load dispatch problem, Department of Electrical Engineering, IIT, Delhi, India, B.K. Panigrahi et al. / Energy Conversion and Management 52 1334–1342.

[12] B. Niu, et al., Multi-objective. (2012). Bacterial Foraging Optimization, Neurocomputing, neucom. 2012.01.044.

[13] David M. Tate and Alice E. Smith. (1992). A Genetic Approach to the Quadratic Assignment Problem, Accepted for publication in Computers & Operations Research.

[14] P. Ji Yongzhong Wu Haozhao Liu. (2006). A Solution Method for the Quadratic Assignment Problem (QAP), The Sixth International Symposium on Operations Research and Its Applications (ISORA'06) Xinjiang, China, August 8–12, ORSC & APORC§pp. 106–117.

[15] Rainer E. Burkhard, Stefan E. Karisch, and Franz Rendli. (1997). Qaplib a quadratic assignment problem library. *Journal of Global Optimization*, 10.

[16] Alfonsas Misevičius, Bronislovas Kilda. (2005). COMPARISON OF CROSSOVER OPERATORS FOR THE QUADRATIC ASSIGNMENT

PROBLEM, ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, Vol.34, No.2.

[17] Tabitha James, Cesar Rego, Fred Glover. (2009). A cooperative parallel tabu search algorithm for the quadratic assignment problem, European Journal of Operational Research 195, 810–826.

[18] Fred Glover (1990). "Tabu Search: A Tutorial". Interfaces.

[19] S.Saha. (2013. Unsupervised classification, similarity measure, classical and metaheuristic approaches, and applications. ISBN: 978-3-642-32450-5.

[20] Jing Dang, Anthony Brabazon, Michael O'Neill, and David Edition. (2008). "Option Model Calibration using a Bacterial Foraging Optimization Algorithm".

[21] G.Naresh, M.Ramalinga Raju and S.V.L.Narasimham. (2011). Bacterial Foraging Algorithm for the Robust Design of Multimachine Power System Stabilizer, International Conference on Signal, Image Processing and Applications With workshop of ICEEA.

[22] R. Vijay. (2012). Intelligent Bacterial Foraging Optimization Technique to Economic Load Dispatch Problem, International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-2.

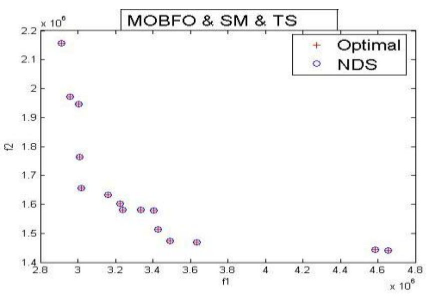
[23] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. (2002). *A fast and elitist multiobjective genetic algorithm: Nsga-ii. Evolutionary Computation*, IEEE Transactions on, 6(2):182 -197, apr.

[24] Fogel, D. B. (1990).AParallel Processing Approach to aMultiple Traveling Salesman Problem Using Evolutionary Programming. In Canter, L. (ed.) Proceedings on the Fourth Annual Parallel Processing Symposium, 318–326. Fullterton, CA.

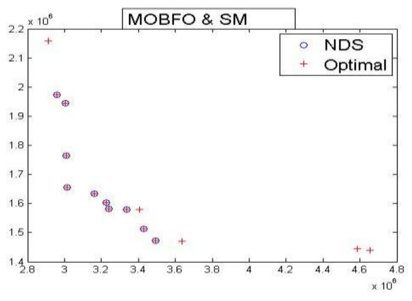
[25] Banzhaf, W. (1990). The "Molecular" Traveling Salesman. Biological Cybernetics 64: 7–14.

[26] P. LARRAN˜ AGA, C.M.H. KUIJPERS, R.H. MURGA, I. INZA and S. DIZDAREVIC. (1999). Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators, Artificial Intelligence Review 13: 129–170.

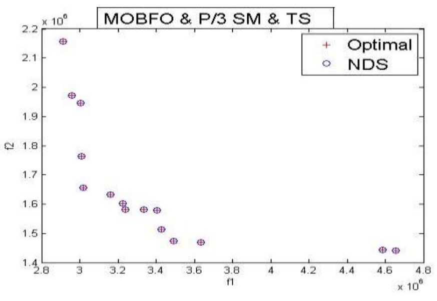
APPENDIX



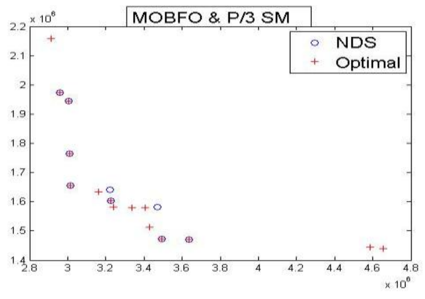
(a)



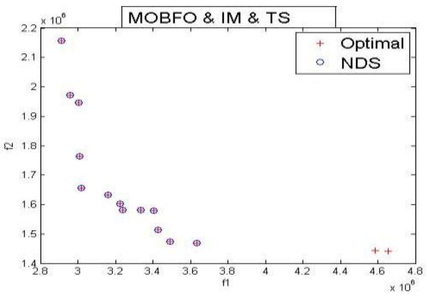
(b)



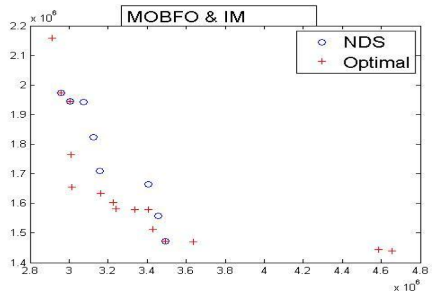
(c)



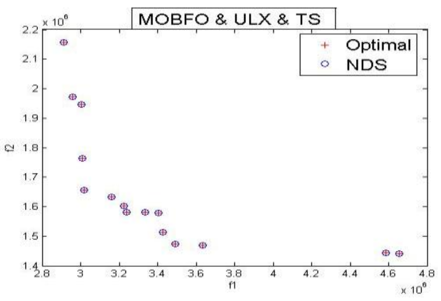
(d)



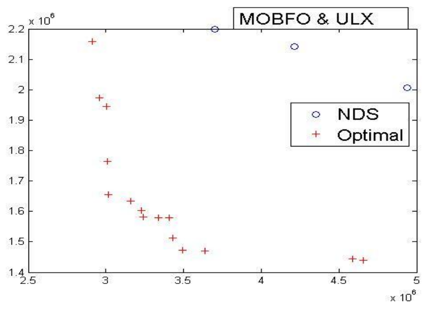
(e)



(f)

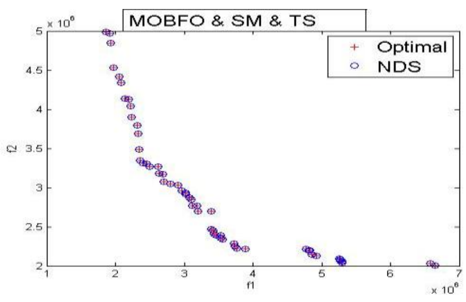


(g)

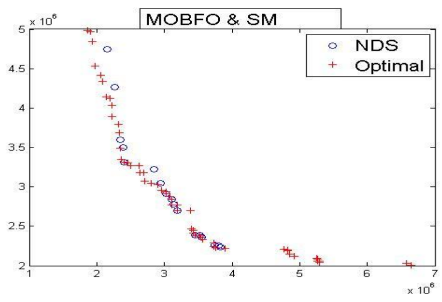


(h)

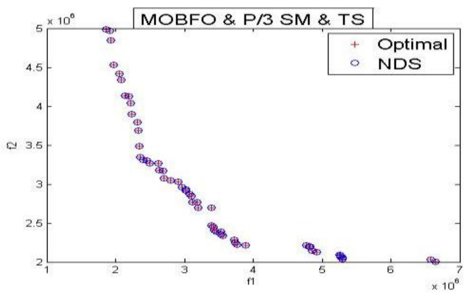
Figure 12: Problem KC10-2fl-2rl with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



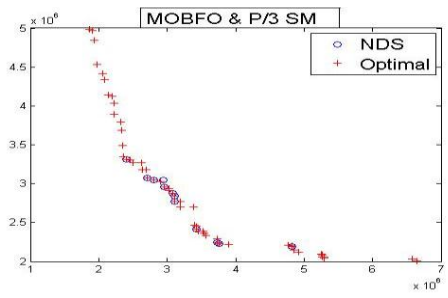
(a)



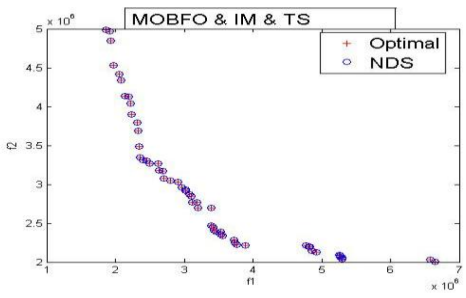
(b)



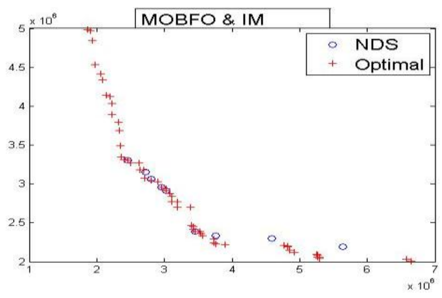
(c)



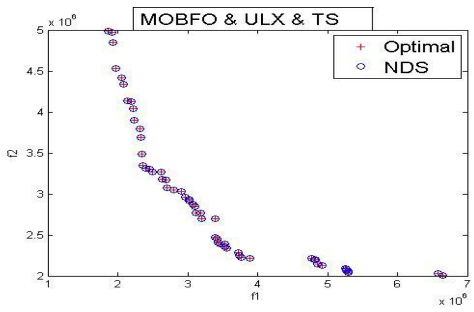
(d)



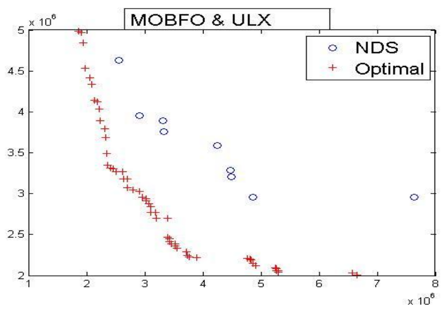
(e)



(f)

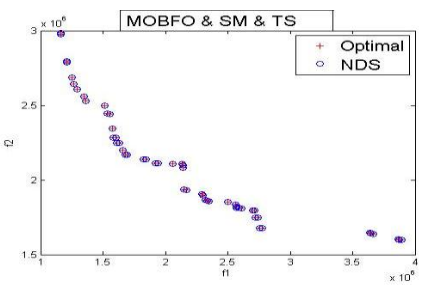


(g)

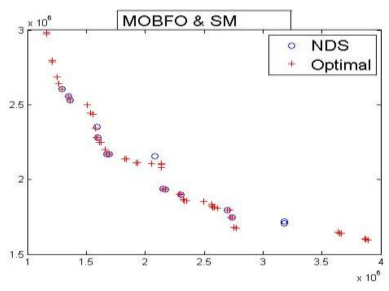


(h)

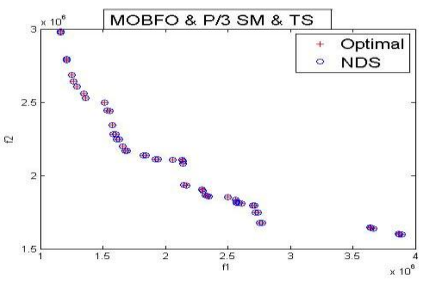
Figure 13: Problem KC10-2fl-3rl with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



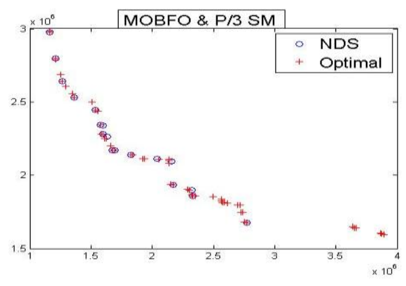
(a)



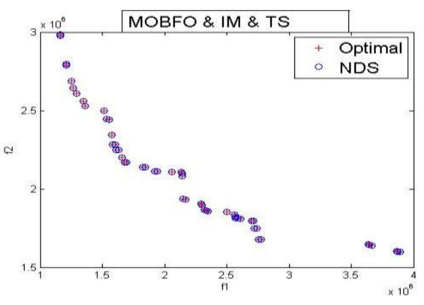
(b)



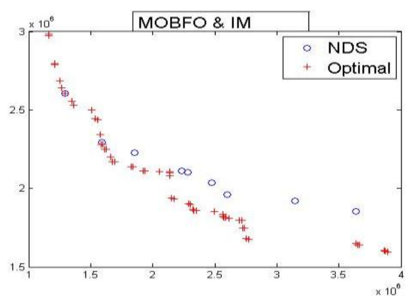
(c)



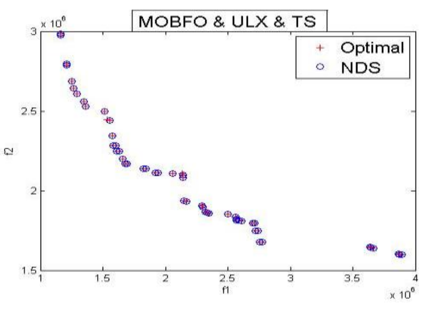
(d)



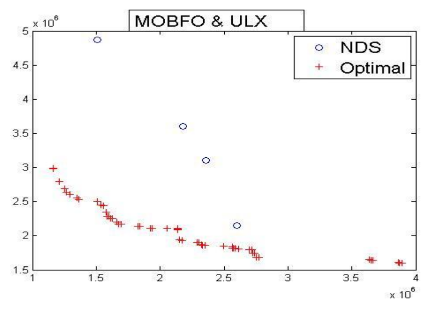
(e)



(f)

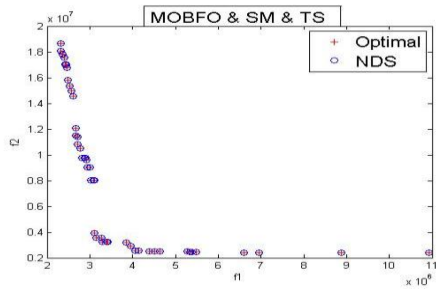


(g)

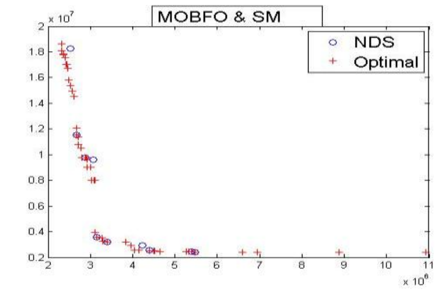


(h)

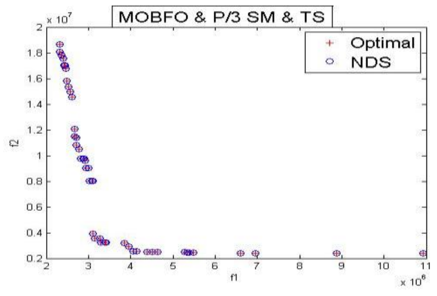
Figure 14: Problem KC10-2fl-4rl with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



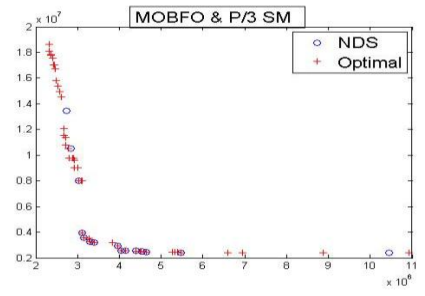
(a)



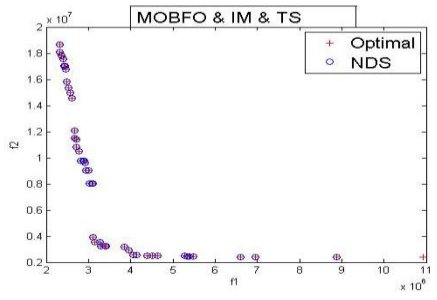
(b)



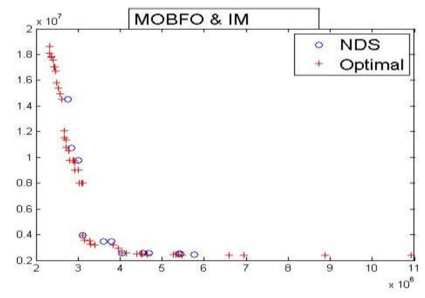
(c)



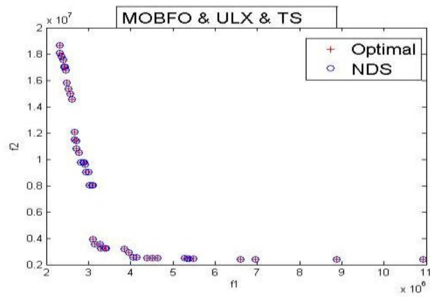
(d)



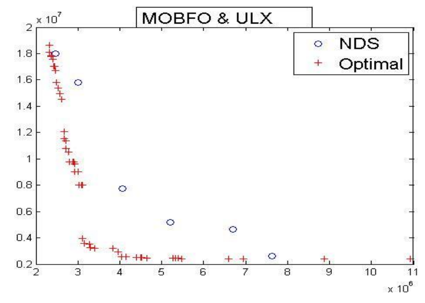
(e)



(f)

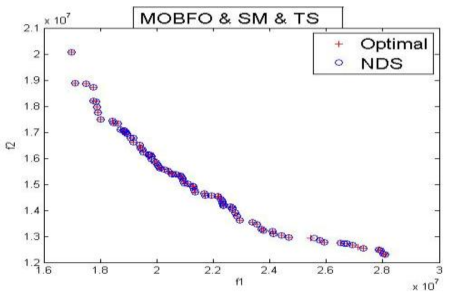


(g)

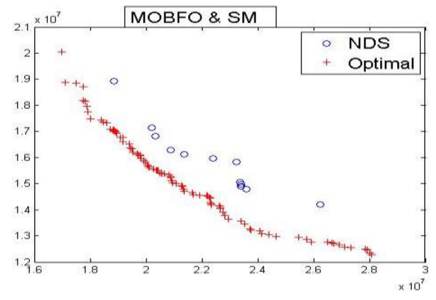


(h)

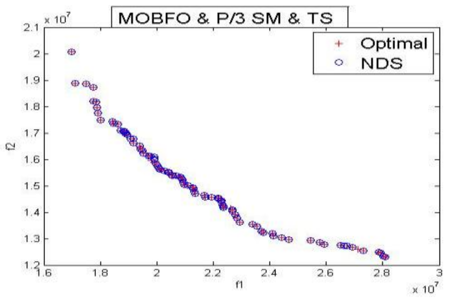
Figure 15: Problem KC10-2fl-5rl with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



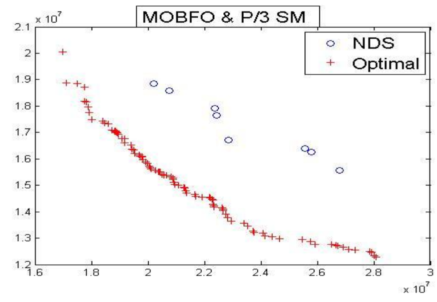
(a)



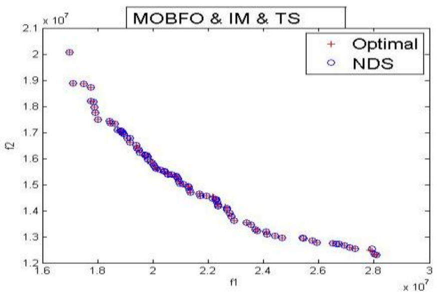
(b)



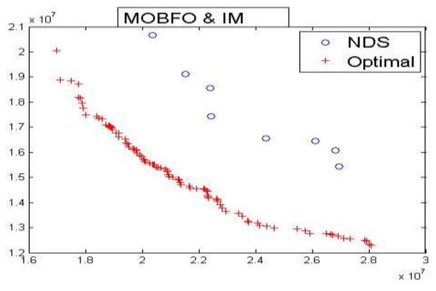
(c)



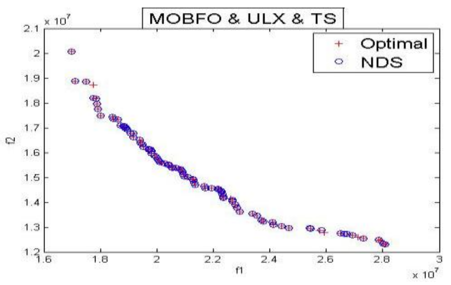
(d)



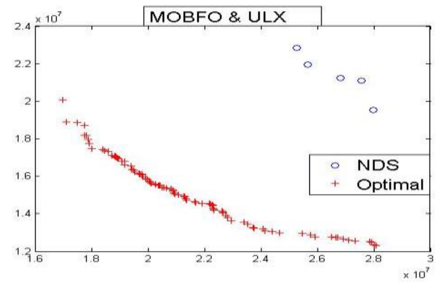
(e)



(f)

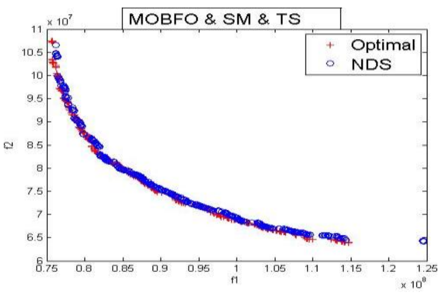


(g)

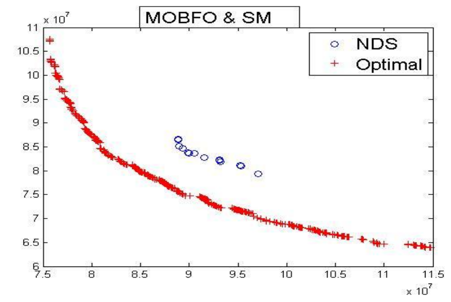


(h)

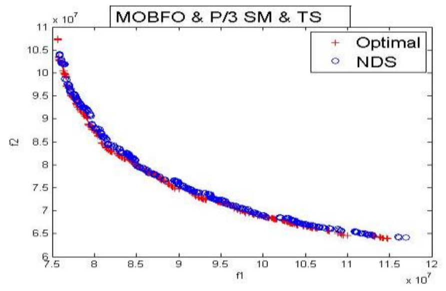
Figure 16: Problem KC20-2fl-1rl with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



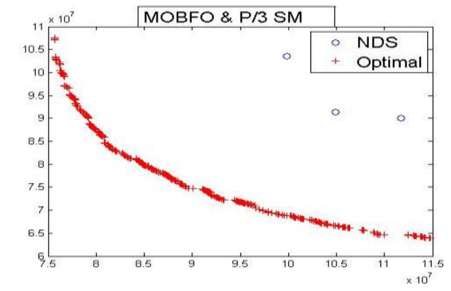
(a)



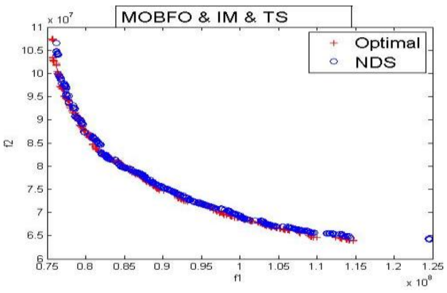
(b)



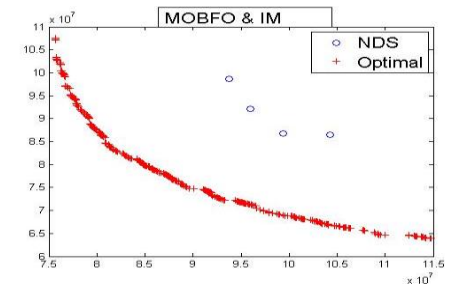
(c)



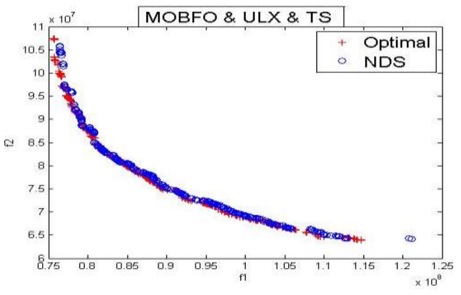
(d)



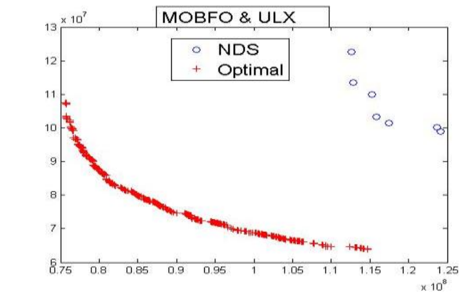
(e)



(f)

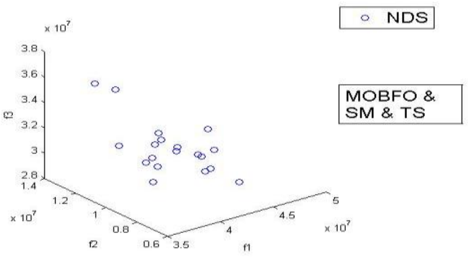


(g)

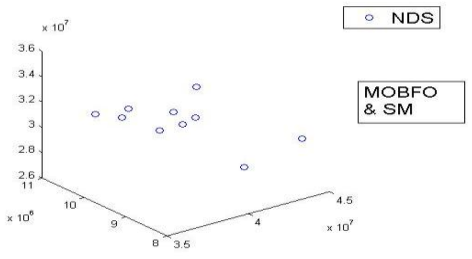


(h)

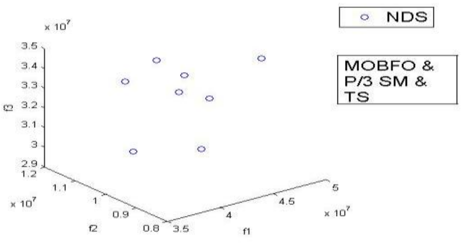
Figure 17: Problem KC50-2fl-1rl with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



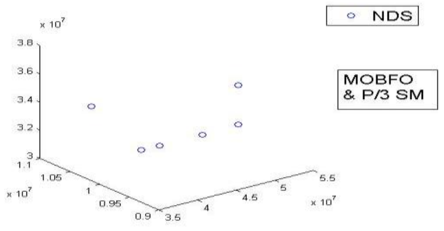
(a)



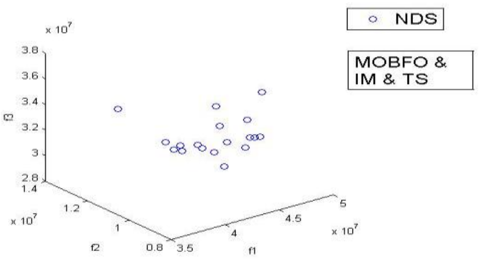
(b)



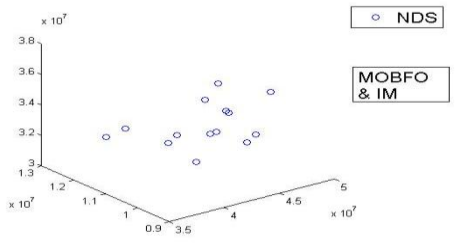
(c)



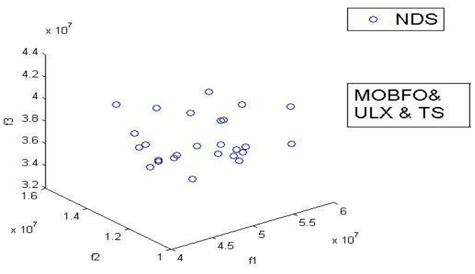
(d)



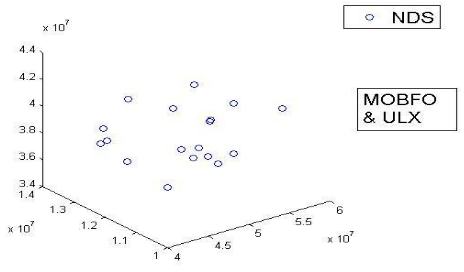
(e)



(f)

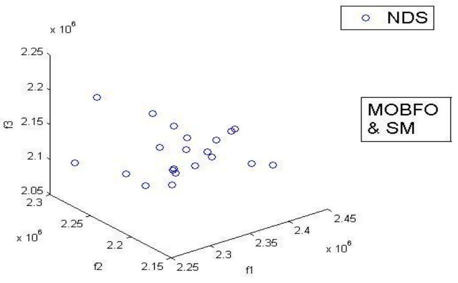


(g)

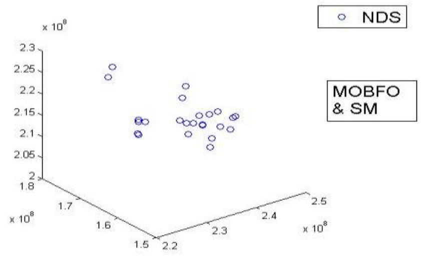


(h)

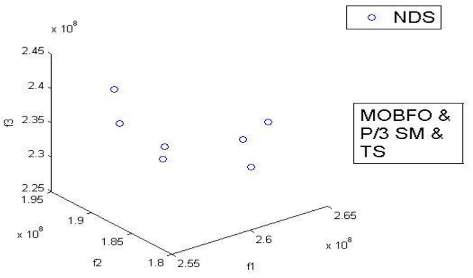
Figure 18: Problem KC30-3fl-1rl with Three Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



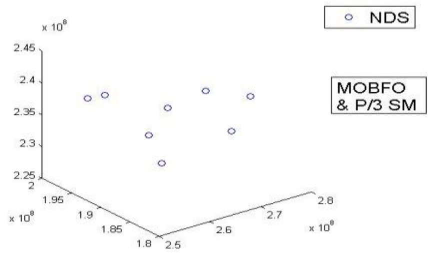
(a)



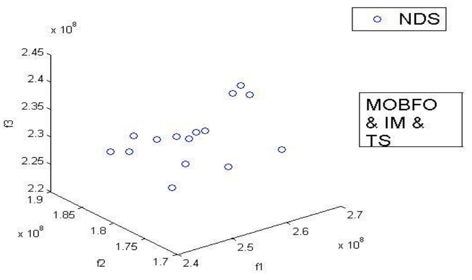
(b)



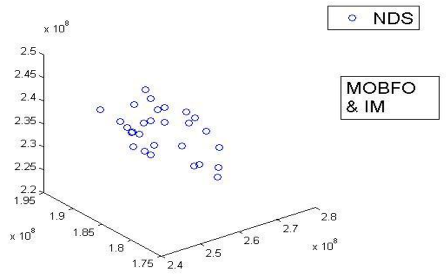
(c)



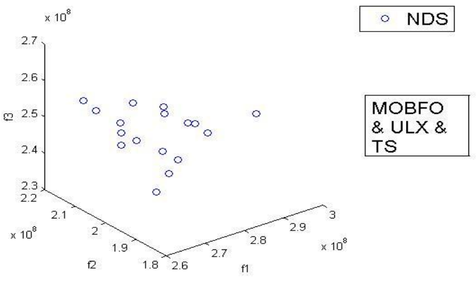
(d)



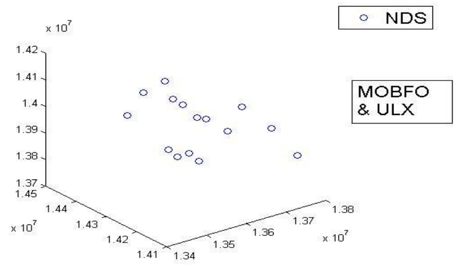
(e)



(f)

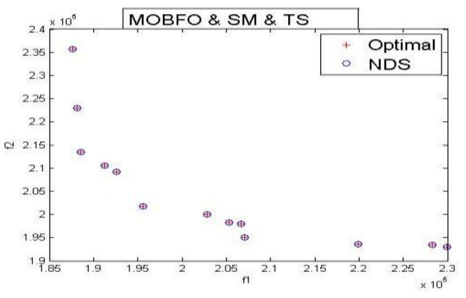


(g)

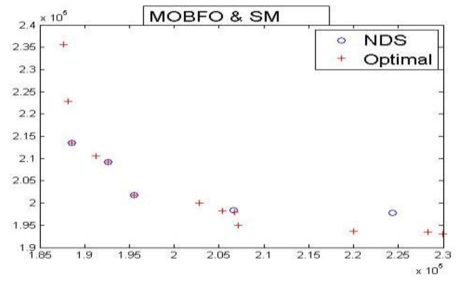


(h)

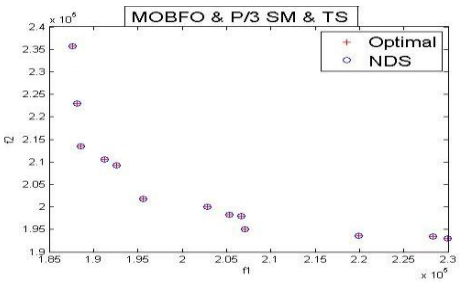
Figure 19: Problem KC75_3fl_1rl with Three Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



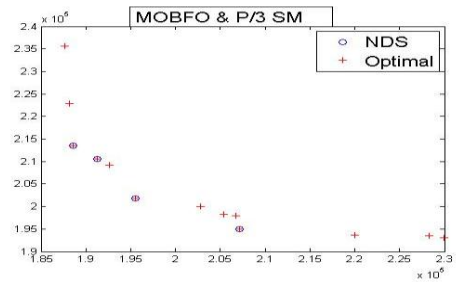
(a)



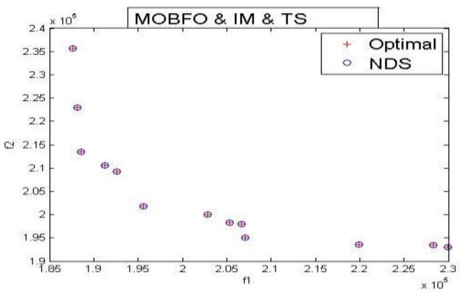
(b)



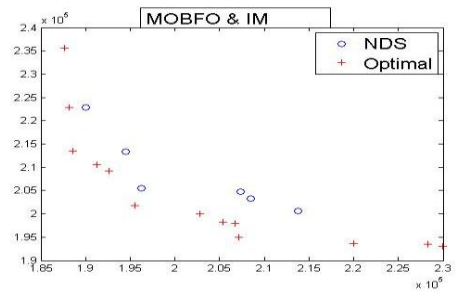
(c)



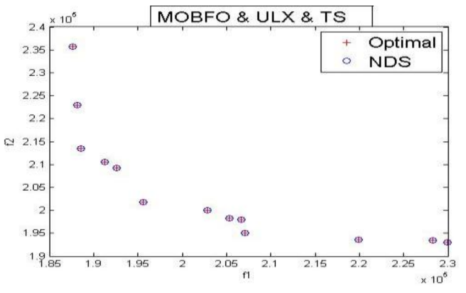
(d)



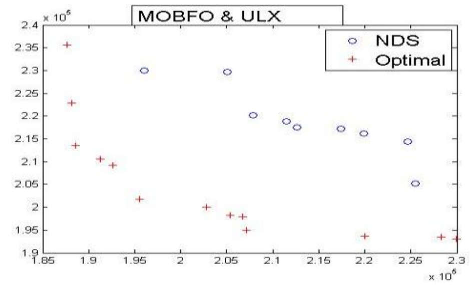
(e)



(f)

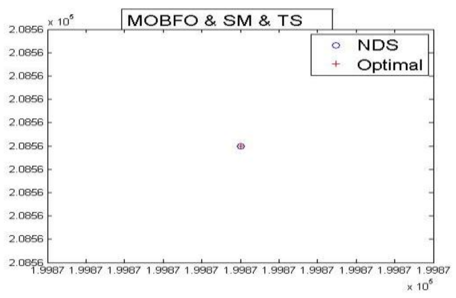


(g)

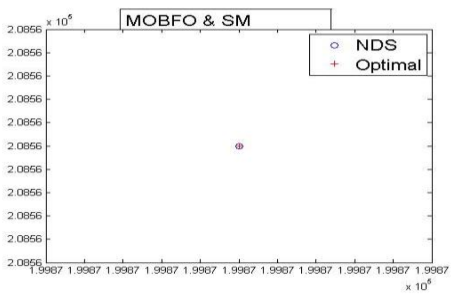


(h)

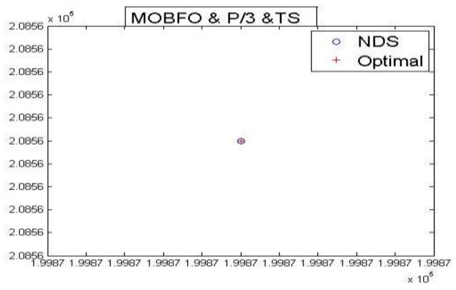
Figure 20: Problem KC10-2fl-1uni with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



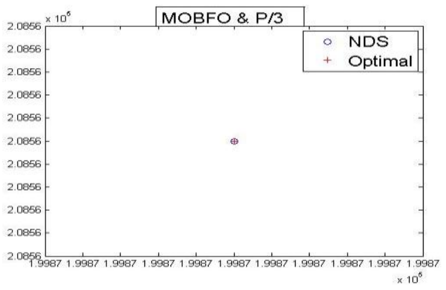
(a)



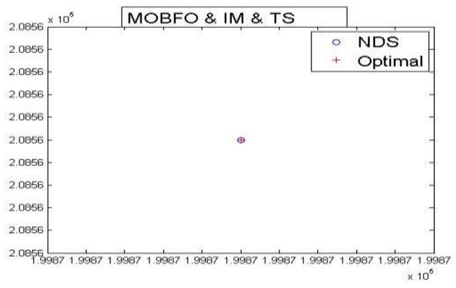
(b)



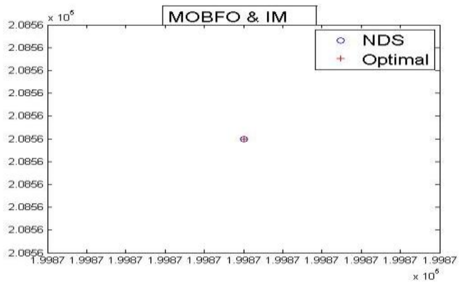
(c)



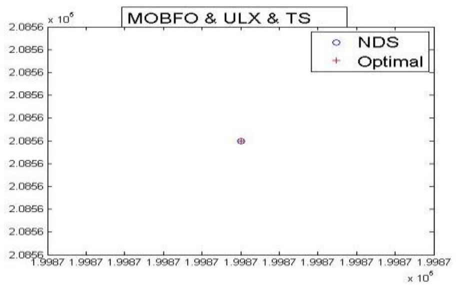
(d)



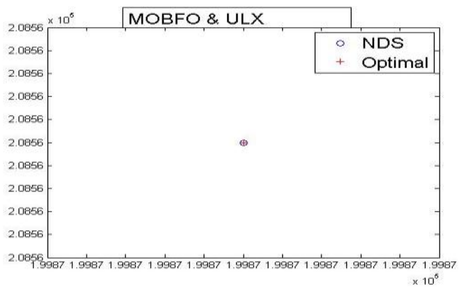
(e)



(f)

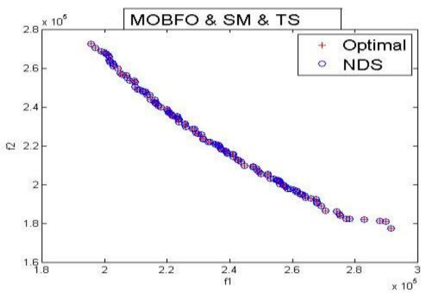


(g)

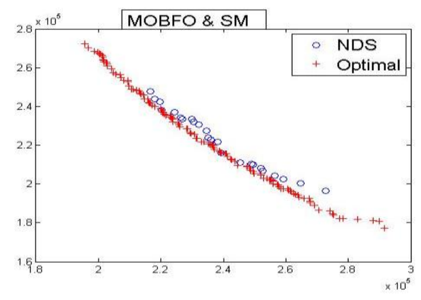


(h)

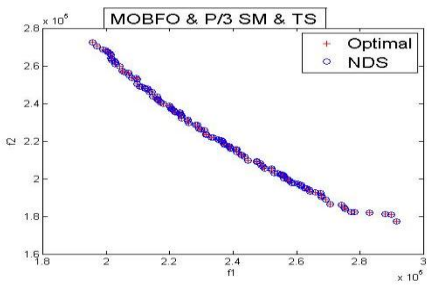
Figure 21: Problem KC10-2fl-2uni with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



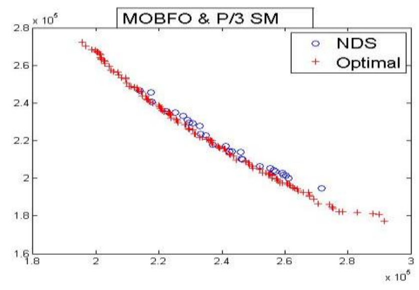
(a)



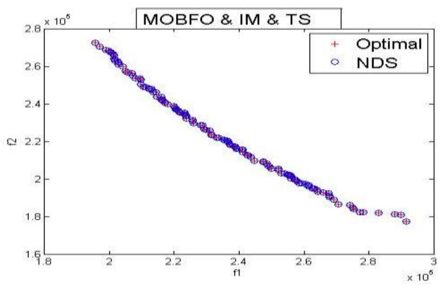
(b)



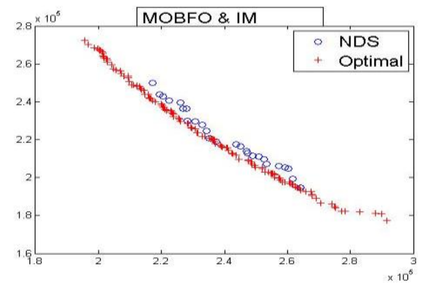
(c)



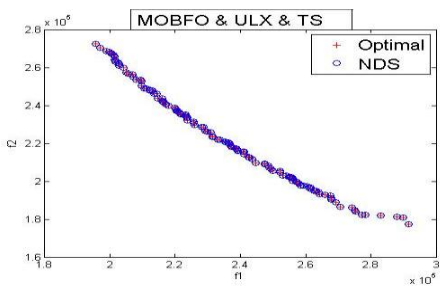
(d)



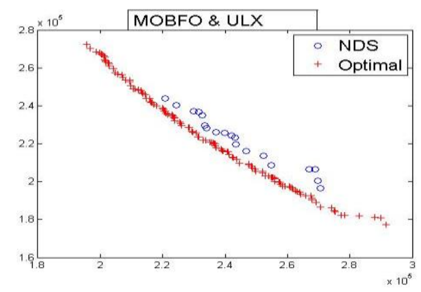
(e)



(f)

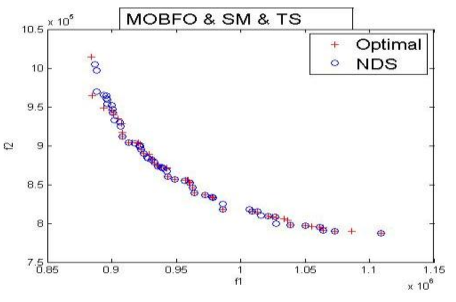


(g)

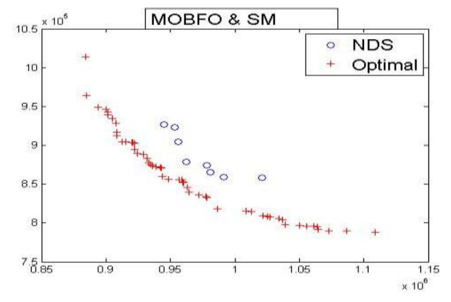


(h)

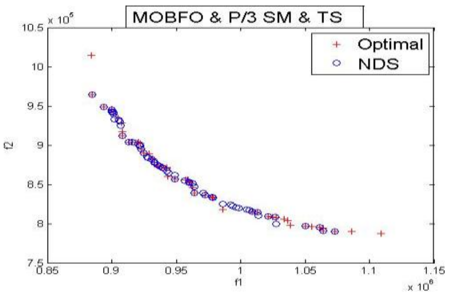
Figure 22: Problem KC10-2fl-3uni with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX



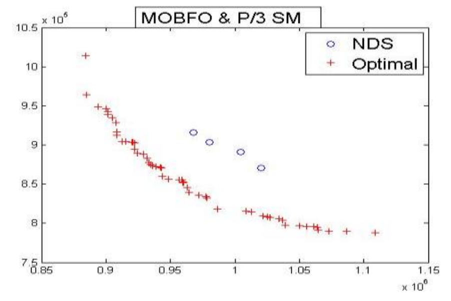
(a)



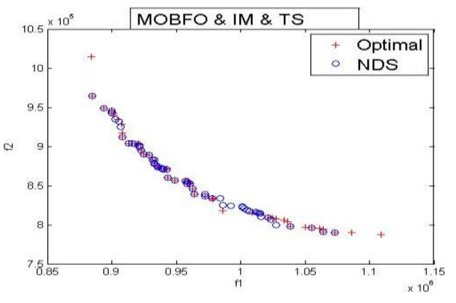
(b)



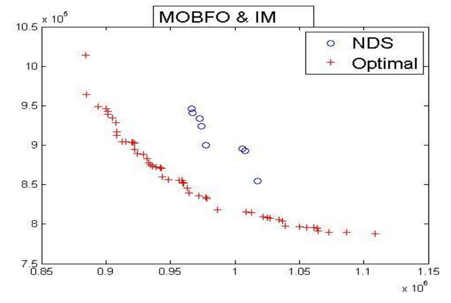
(c)



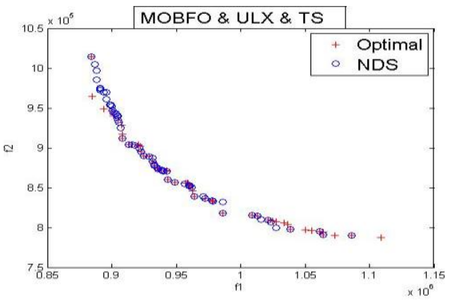
(d)



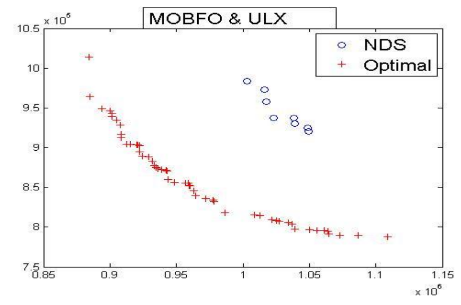
(e)



(f)



(g)



(h)

Figure 23: Problem KC20-2fl-1uni with Two Objectives: a) MOBFO & swap mutation & tabu search, b) MOBFO & swap mutation, c) MOBFO & P/3 mutation & tabu search, d) MOBFO & P/3 mutation, e) MOBFO & inversion mutation & tabu search, f) MOBFO & inversion mutation, g) MOBFO & ULX & tabu search, h) MOBFO & ULX

Table 3: ER and HV for Figure 12

KC10-2fl-2rl	Error Ratio	Hyper volume
Swap mutation	0	0.11355
Shift mutation	0	0.10800
p/3 swap	0	0.10830
ULX	0	0.11110

Table 4: ER and HV for Figure 13

KC10-2fl-3rl	Error Ratio	Hyper volume
Swap mutation	0	0.35860
Shift mutation	0	0.34480
p/3 swap	0	0.35860
ULX	0	0.35165

Table 5: ER and HV for Figure 14

KC10-2fl-4rl	Error Ratio	Hyper volume
Swap mutation	0	0.24745
Shift mutation	0	0.24115
p/3 swap	0	0.25220
ULX	0	0.24765

Table 6: ER and HV for Figure 15

KC10-2fl-5rl	Error Ratio	Hyper volume
Swap mutation	0	0.63800
Shift mutation	0	0.63650
p/3 swap	0	0.64565
ULX	0	0.64135

Table 7: ER and HV for Figure 16

KC20-2fl-1rl	Error Ratio	Hyper volume
Swap mutation	0.0337	0.10805
Shift mutation	0.0659	0.10595
p/3 swap	0.0556	0.10505
ULX	0.0556	0.10065

Table 8: ER and HV for Figure 17

KC50-2fl-2rl	Error Ratio	Hyper volume
Swap mutation	0	0.10510
Shift mutation	0.9924	0.18100
p/3 swap	1	0.12655
ULX	1	0.15510

Table 9: ER and HV for Figure 20

KC10-2fl-1uni	Error Ratio	Hyper volume
Swap mutation	0	0.03090
Shift mutation	0	0.02840
p/3 swap	0	0.03045
ULX	0	0.03035

Table 10: ER and HV for Figure 21

KC10-2fl-2uni	Error Ratio	Hyper volume
Swap mutation	0	0
Shift mutation	0	0
p/3 swap	0	0
ULX	0	0

Table 11: ER and HV for Figure 22

KC10-2fl-3uni	Error Ratio	Hyper volume
Swap mutation	0	0.06760
Shift mutation	0	0.06775
p/3 swap	0	0.06940
ULX	0	0.066605

Table 12: ER and HV for Figure 23

KC20-2fl-1uni	Error Ratio	Hyper volume
Swap mutation	0.5926	0.03575
Shift mutation	0.5862	0.03570
p/3 swap	0.6271	0.03495
ULX	0.5522	0.03550