Game Theoretic Personality Changing MOPSO method for Multi-Objective Optimization

Mawadavilage Malith Madhushanka , Index no - 13000675

University of Colombo

School of Computing

2017

Game Theoretic Personality Changing MOPSO method for Multi-Objective Optimization

M.G.M Madhushanka

Index No: 13000675

Supervisor: Dr. D.N. Ranasinghe

Co-Supervisor: Dr. T. Sritharan

December 2017

Submitted in partial fulfillment of the requirements of the B.Sc in Computer Science Final Year Project (SCS4124)

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: M. G. M Madhushanka

……………………………………………

Signature of Candidate   Date:

This is to certify that this dissertation is based on the work of

Mr. M. G. M Madhushanka
under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Supervisor Name:

……………………………………………

Signature of Supervisor   Date:

Abstract

Multi-objective particle swarm optimization is an extension/generalization of the particle swarm optimization for optimizing more than one objective simultaneously. Since particle swarm optimization is inspired by animal behavior, mainly bird flocking and fish schooling, and also the competitive nature among objectives which can be observed in multi-objective optimization, it is intuitive to look into this problem from a game theoretic point of view.

In this thesis we have introduced a novel method called Personality changing multi-objective particle swarm optimization (PC-MOPSO) which is based on game theory and is an enhancement to standard-MOPSO [1]. We apply PC-MOPSO to the specific problems of two-objective optimization. In PC-MOPSO each particle has two personalities and each personality is trying to maximize its payoff by making a rational decision/choice. As a result of this iterative process the final optimal solutions are achieved.

Four standard test functions used for evaluation and simulation of results show that the proposed method is capable of handling ZDT1 problem with 2.79% higher accuracy than the standard-MOPSO and also performed competitively in other problems. One interesting observation which is unique to the new method is, unlike in other MOPSO methods which have utilized an external repository to find final solutions, in this method solutions are achieved in a distributed manner. This can be seen as similar to most evolutionary algorithms.

Preface

As a basis and the starting point of this research Coello Coello’s “handling multi-objectives with particle swarm optimization” is considered. And for the implementation purposes Yarpiz’s Matlab implementation of above method is used as the basis and modified according to requirements. Further for the evaluation purposes Johann’s implementation of hypervolume indicator metric is used. All new concepts implementation and analysis carried out were original work.

Acknowledgement

I would like to express my sincere gratitude to my research supervisor, Dr. D.N. Ranasinghe, a Senior Lecturer of University of Colombo School of Computing and my research co-supervisor, Dr. T. Sritharan, Senior Lecturer of University of Colombo School of Computing for providing me continuous guidance and supervision throughout the research.

Contents

List of Figures

List of Tables

List of Acronyms

PSO –	Particle Swarm Optimization
MOPSO –	Multi-Objective Particle Swarm Optimization
MOO/MO –	Multi-objective Optimization
DE -	Differential-evolution
GT –	Game Theory
PC-MOPSO -	Personality changing Multi-Objective PSO

Chapter 1 - Introduction

Optimization is a branch of mathematics and numerical analysis. Almost all problems in other fields can be formulated as optimization problems or search problems. Some of these problems (simple problems) can be solved by traditional methods based on mathematical analysis, but most of the problems are too hard to solve by traditional methods due to complexity and need new approaches to solve.

Nature is a system of vast complexity and natural computing is the way of using those nature inspired methods for our advantage to solve those hard problems. Evolutionary computing, neural computing, cellular automata, swarm intelligence, quantum computing are some of the well-known examples for natural computing.

In this project, we are only concerned with very specific area on particle swarm optimization on solving multi-objective optimization problems in cooperating with game theoretic ideas. There are various heuristic methods, which can be found on solving multi-objective optimization problems which are mainly based on evolutionary algorithms and genetic algorithms but almost all of them are very complex and have their own weaknesses such as low efficiency and low speed. But recently, due to the popularity, simplicity and efficiency of the Particle Swarm Optimization (PSO) method, researchers have proposed extending single objective PSO to multi-objective optimization which is called MOPSO to overcome those weaknesses and also as a novel approach which is capable of solving MO problems. Several studies can be found on this area and they showed its superiority compared to existing methods. Although this looks promising, as same as other heuristic methods MOPSO has several problems such as premature convergence, robustness, speed, accuracy etc.

1.1 Background to the Research

Particle swarm optimization (PSO), invented by Kennedy and Eberhart in 1995 [2], is a very popular nature-inspired meta-heuristic algorithm for solving optimization problems which is inspired from studies of synchronous bird flocking and fish schooling. It basically reflects social interaction of individuals living together in groups. for that it uses notion of employing many autonomous particles that can act together in simple ways to produce complex behaviors. It abstracted each member/bird as a particle and they work under social behavior in swarm and try to find global best solution by adjusting attributes of each particle according to its personal best and global best of entire swarm at each iteration.

There are several types of advancements can be found on PSO and they can be mainly categorized in to,

1. Modification of PSO – Quantum behavior based PSO [3], chaotic PSO [4], fuzzy PSO [5]
2. Hybridization of PSO – Cellular PSO [6], artificial immune system (AIS) [7], Tabu-search [8]
3. Extension of PSO – multi objective, discrete, binary optimization
4. Theoretical analysis of PSO – convergence analysis
5. Parallel implementation of PSO

In this project, we are focusing on domain, Extension of PSO for multi-objective optimization problems. Several approaches have already been proposed in this category. Our approach is to looking at MOPSO in a game theoretic perspective
and also, to add possible enhancements.

1.2 Research Problem / Hypothesis

PSO as a swarm based evolutionary heuristics has been successfully used in single objective optimization. For multi-objective optimization, work carried out shows the usefulness of Multi-Objective PSO concept. We hypothesize that game theoretically interacting particles in a PSO are better for multi-objective optimization and has interesting effects.

1.3 Justification for the research

Multi Objective Optimization is a very important area in terms of both theoretical and practical aspects. Most of the real-world problems have this conflicting multi-objective nature inherently, in that cases there is no single solution exist and we have to go for approximate solutions which typically have qualities like robustness, accuracy, diversity etc. Evolutionary algorithms and genetic algorithms are considered as traditional methods for tackling MOO problems. But the burst of popularity of the Particle Swarm Optimization algorithm for single objective optimization attract many researchers to extending PSO methods for solving MOO problems and number of publications have proved its worthiness. Most of these researches were focused on small tweaking of original MOPSO which can ultimately improve effectiveness of the algorithm.

Since PSO is a social behavior inspired method which involves interactions of particles and the conflicting nature of each objectives in MOO, it is rational and logical to look at this whole picture from more behavioral point of view, more formally game theoretic view which may involves concepts like competition and cooperation (more natural way) or other game theory mechanics. There are several researches can be found based on these facts and they also shows that their methods can be superior to other methods.

So, in this research our focus is to investigate possible game theoretic models, additions for the traditional MOPSO algorithm which may leads to improvement over traditional methods.

1.4 Methodology

Since this research is about utilizing the behavior of particles of swarm for solving multi-objective optimization, simulation methodology was used as the main methodology. In order to evaluate purposed MOPSO method's performance, computational experiments were performed over new purposed method and standard general MOPSO. And compared them using several standard performance indicators. MATLAB programming environment was used for implementing both algorithms and for all the evaluation and comparisons.

1.5 Outline of the Dissertation

In the second chapter, basic concepts behind the topic are layed out. Definitions of single objective optimization problem, general multi-objective optimization problem, pareto dominance, pareto optimality given too. Then conventional particle swarm optimization algorithm and multi-objective optimization algorithm are discussed. Then existing approaches to MOPSO is introduced and Game theory based MOPSO and other hybrid MOPSO methods discussed and conclusions are provided.

In chapter three, new MOPSO method named Personality changing MOPSO is presented and algorithm is given and discussed.

In chapter four detailed implementation of the algorithm is provided with MatLab codes.

In chapter five simulation results are given and discussed. And in finally in chapter six conclusions of the thesis, limitations and possible future research areas are given.

1.6 Delimitations of Scope

In this research, we are only focusing on two-objective unconstrained optimization problems. Convex optimization problems are omitted and focusing only on non-convex optimization. And also, we are using measurement tools which used in previous researches for comparison purposes. Theoretical analysis of the purposed method is also not discussed. And for the evaluation we use popular algebraic two-objective problems which was used in previous researches on this domain.

1.7 Conclusion

This chapter laid the foundations for the dissertation. It introduced the research problem, research questions and the hypotheses. Then the research was justified, definitions were presented, the methodology was briefly described, the dissertation was outlined, and the limitations were given. On these foundations, the rest of the dissertation proceed with more detailed description of the research.

Chapter 2 - Literature Review

2.1 Introduction

Multi – Objective Optimization is an area of multiple criteria decision making, which involves with optimizing multiple objective functions simultaneously. We can see practical applications of multi – objective optimization in various fields including engineering, economics and logistics where optimal decision needs to be taken in the presence of trade-offs between two or more conflicting objectives. Minimizing the cost while maximizing comfort when buying car, minimizing energy consumption while maximizing performance when building machine are some examples of multi objective optimization usage in the practical scenarios.

Apart from the trivial problems which there exist single solution which simultaneously satisfy all objective functions optimally, for the most of the problems there is no single solution exists. In such cases we said that their objective functions are conflicting and there exists number of pareto optimal solutions.

Pareto optimality is a concept which is used in game theory which was introduced by Italian economist, mathematician Vilfredo Pareto as a measure of efficiency. Outcome of a game is pareto optimal if there is no other outcome that makes every player at least as well-off and at least one player strictly better-off. In other words, outcome cannot be improved without hurting at least one player. This concept is borrowed in various fields such as economics and applied mathematics / computational mathematics for handling those conflicting issues.

In multi – objective optimization domain, a solution is called pareto optimal if it is nondominated, if none of objective functions can be improved in some value without degrading some other objective values. And other important fact is those pareto optimal solutions are said to be equally good in mathematical sense without considering any additional subjective preference information which can be used to choose most suitable solution from all pareto optimal solution set.

Goal of the multi – objective optimization can be vary depending on different philosophical viewpoint but in general goal or the objective is to find all pareto optimal solutions. If we look further on these areas we can categorize multi – objective optimization in to two different categories

1. Non – preference methods

2. Preference methods

Here, non – preference methods are focusing on finding all pareto optimal solutions and Preference methods are further improving this by using preference info to find optimal solution which is most suitable for the relevant requirement.

Evolutionary algorithms are considered as the standard approach to solving MOO problems and it was well established due to their ability of finding pareto optimal sets. Although they were successful on that there are some disadvantages. The main disadvantage is its computational cost and time, this is a common problem in evolutionary methods due to inherent nature. So, researchers started to look for alternative methods which are quick and efficient.

With the introduction of Particle Swarm Optimization algorithm in 1995 by Kennedy and Eberhart [2] which was inspired by bird flocking behavior and their series of algorithmic variances and extensions showed that simple but efficient methods can be utilized to solve single optimization problem, effectively with less computational cost. It also provided some interesting insight to formalize animal and intelligent behavior in the field of artificial intelligence and sociology. PSO soon became very famous among researchers and lot of studies conducted to further development and theoretical analysis.

As a result of these development and extensions, Multi – Objective Particle Optimization was first introduced by Coello Coello, 2002 [9] as a generalization of PSO for multi-objective optimization. The idea is to utilize the advantages they got in single objective optimization for solving multi – objective optimization problems. Approaches which were taken for MOPSO can be divide in to two main categories depending on how they handle their objective functions that are needed to be optimized.

1. The first category consists of unification or reduction of all of the objective functions which are needed to be optimized in to single objective function with multiple variables and then using modified single objective PSO for finding pareto optimal solution set.
2. Second category consists of approaches that consider each objective function separately

By looking at the swarm behavior in real world we can see much complex decision making is happening between individuals and as well as between groups. They exhibit competitive and cooperative behavior depending on the context and also selfish, and generous behaviors, to maximize their gains to achieve some goal as a group and also what they trying to achieve also can be vary. In other words, we can say that they have multiple objective goals. By considering those facts it is intuitive to think that there must be some involvement of multi – objective optimization as well as game theory in here. More accurately it is logical and intuitive to model swarm metaphor using game theory to achieve multi – objective optimization using PSO.

In the rest of the chapter we are going to focus on reviewing existing literature on multi – objective optimization and their modified versions that are involving game theory or similar behavior mechanism as an extension to general MOPSO methods.

2.2 Definitions

Definition 1 (Single Objective Optimization Problem):

"A single objective optimization problem is defined as minimizing or maximizing of objective $f(x)$, subject to $g_i(x) \leq 0,\ i = 1,2,3, \dots, m$ and $\hbar_j(x) = 0,\ j = 1,2,3, \dots, p$; where $x \in S \subset \mathbb{R}^n$ and S is the solution space (decision variable). A solution minimizes or maximizes the scalar $f(x)$, where n-dimensional decision variable *vector* $x = (x_1, x_2, x_3, \dots, x_n)$".

Definition 2 (A General MOOP):

Multi-Objective Optimization problem is defined as minimizing/maximizing $F(x) = (f_1(x), f_2(x), f_3(x), \dots, f_n(x)$, subject to $g_i(x), \leq 0,\ i = 1,2,3, \dots, m$ and $h_k(x), = 0,\ k = 1,2,3, \dots, p;\ x \in S$. An MOOP solution minimizes/maximizes the components of a vector $F(x)$ where $x = (x_1, x_2, x_3, \dots, x_n)$ is n-dimensional decision variable from some universe S. It is denoted that $g_i(x) \leq 0$ and $\hbar_k(x) = 0$ represents the constraints that must be fulfilled while minimizing/maximizing $F(x)$, and S contains all possible x that can be used to satisfy an evaluation of $F(x)$.

Definition 3 (Pareto dominance): The vector u is said to dominate vector v, if and only if it holds that:

$u_i \leq v_i$, for all $i = 1, 2, \dots, k$, and, $u_i < v_i$, for at least one component i.

Definition 4 (Pareto optimality): A solution, $x \in A$, of the MO problem is said to be Pareto optimal, if and only if there is no other solution, $y \in A$, such that $f(y)$ dominates $f(x)$. Alternatively, we can say that x is nondominated with respect to A. The set of all Pareto optimal solutions is called Pareto optimal set and will be henceforth denoted as P^*.

2.3 Particle Swarm Optimization (PSO)

This method was first introduced by Kennedy and Eberhart in 1995 [2] as result of a mathematical investigation on animal behavior mostly based on bird folks and fish schools. The main ideas were nearest neighbor velocity matching and acceleration by distance. These ideas were formulated into equations and converted in to PSO algorithm. In PSO manipulation of a swarm is different from evolutionary algorithms, because it promotes cooperative behavior rather than competitive model.

Let $f: S \rightarrow \mathbb{R}$ be objective function, S is d-dimension search space, n is number of particles where $S = \{x_1, x_2, x_3, \ldots, x_n\}$. i^{th} particle of the swarm can be represented as $X_i = (x_{i1}, x_{i2}, x_{i3}, \ldots, x_{id}) \in S$ and personal best position pbest$_i$= $(pbest_{i1}, pbest_{i2}, pbest_{i3}, \ldots, pbest_{id}) \in S$. And the velocity of the ith article is $V_i=(v_{i1}, v_{i2}, v_{i3}, \ldots, v_{id})$. The particle movement for $(t+1)^{th}$ iteration updated as follows.

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad \textbf{(1)}$$

$$V_i(t+1) = V_i(t) + c_1 r_{i,1}(t)\,(pbest_i(t) - X_i(t)) + c_2 r_{i,2}(t)\,(gbest(t) - X_i(t)) \quad \textbf{(2)}$$

Where $i = 1,2,3, \ldots, n,$

$X_i(t),$ = position of the ith particle at tth iteration,

$V_i(t),$ = velocity of the ith particle at tth iteration.

gbest(t) = best position founded by entire swarm so far.

pbest(t) = best position founded by each particle.

1. The component (V_i) models the tendency to continue in the same direction.

2. The component ($pbest_i$) is a linear attraction toward the personal best position ever found, which is scaled by random weight $c_1r_{i,1}$.

3. The third component ($gbest_i$) is a linear attraction towards the global best position found by any particle of the swarm, which is scaled by another random weight $c_2r_{i,2}$.

Original PSO model was tend to be trapped in local minima due to heavy dependent nature of the one global best so to avoid that later neighborhood topologies introduced. The following are the most common adapted topologies:

1. Fully Connected Graph: In this topology, all members of the given swarm are connected to one another.
2. Star Network: In this topology, one particle, called the focal particle, is connected to all the remaining particles in the swarm, but each is connected to that one only. It is also known as wheel topology
3. Tree Network: All the particles are arranged in a tree structure in which each node has exactly one particle [61]. It is also known as hierarchical topology

Overall procedure of PSO can be shown in following algorithm

```
1:   begin
2:       for each particle of the swarm
3:           Initialize particles position and velocity randomly
4:       end for
5:       do
6:       for each particle of the swarm
7:           Evaluate the fitness function
8:           if the objective fitness value is better than the personal best
             objective fitness value (pbest) in history Current fitness value set as
             the new personal best (pbest)
9:           end if
10:      end for
11:      From all the particles or neighborhood, choose the particle with best fitness
         value as the gbest or lbest
12:      for each particle of the swarm
13:          Update the particle velocity according to Eq. 2
14:          Update the particle position according to Eq. 1
15:      end for
16:  until stopping criteria is not satisfied
```

2.4 MOPSO (Multi – Objective PSO)

The first MOPSO concept was introduced by Carlos A. Coello Coello in 2002 [9] as a proposal for multi objective optimization using particle swarm method. In his paper, he proposed a method based on weighted inertia PSO variance [10] which is a standard PSO model for optimizing single optimization problems. The main idea was to maintain a repository or an external archive for storing pareto optimal solutions in each iteration. And also, there was a mechanism implemented for updating / maintaining archive. Also in the archive, they used mathematical structures such as hypercube to sort and store solutions. And the data stored in the archive was used as historical data for updating current equation and make decisions. They tested results of their proposed algorithm with evolutionary multi objective algorithms such as Pareto Achieved Evolution Strategy (PAES) [11] and Non-Dominated Sorting Generic Algorithm II [7] (NSGA II) [12] and showed that MOPSO method is efficient and capable of handling MO problems. In 2004 same authors published a more detailed paper [1] on same topic, further improved by adding mutation operator to the MOPSO and further showed its capabilities by comparing it to existing methods.

By borrowing concept of neighborhood from single PSO variance, MOPSO was also proposed, and in these kind of MOPSO's swarm is divided into groups and some neighborhood topology (figure .1 & figure .2) is established (ring is the most popular method). Instead of depending on every particle here leaders for each group was introduced and decisions were made with considering those values of the leaders [13]. In this approach, the main problem was the method of selecting leaders. And there are several leaders' selection techniques were proposed by using quality measures. Such as

1. Nearest neighbor density estimator [12]

The nearest neighbor density estimator gives an idea of how crowded the closest neighbors of a are given particle, in objective function space. This measure estimates the perimeter of the cuboid formed by using the nearest neighbors as the vertices.

2. Kernel density estimator [14] [15]

When a particle is sharing resources with others, its fitness is degraded in proportion to the number and closeness to particles that surround it within a certain perimeter. A neighborhood of a particle is defined in terms of a parameter called σ share that indicates the radius of the neighborhood. A neighborhood of a particle is defined in terms of a parameter called σ share that indicates the radius of the neighborhood. Such neighborhoods are called niches.

MOPSO methods can be mainly categorized in to five groups.

1. Aggregating approaches [16] [17]

In this approach, all the objectives of the problem transform into single one.

2. Lexicographic ordering [18]

In this approach, they rank objectives in order of importance.

3. Sub-Population approaches [19] [20]

These approaches are involving with several sub populations as single optimizers

4. Pareto-based approaches [21] [22] [9] [1]

These approaches use leader selection techniques based on Pareto dominance.

5. Combined approaches. [23] [24]

Combination of above mentioned methods

2.5 Pareto archive management (General algorithm)

This mechanism is based on pareto dominance concept which originally used in area of game theory. Idea is to find non- dominated solution set. After stopping criteria met the stored non-dominated solutions in the archive is considered as the possible solutions for the original problem. And also, researchers have been using several techniques such as fitness sharing, crowding distance to get a more distributed and diverse solutions which is preferred in MOO.

Following algorithm illustrate the simple and general version of such pareto achieve maintenance procedure.

```
1: function UPDATEARCHIVE (archive, candidate)
2:     for solution $\in$ archive do
3:         if dominates (solution, candidate) then
4:             return False
5:         end if
6:         if dominates (candidate, solution) then
7:             remove (archive, solution)
8:         end if
9:     end for
10:    append (archive, solution)
11:    return True
12: end function
```

By going through previous researches, we can see that there are lot of researches focused on application of MOPSO for solving practical and other specialized problems.

Dhafar Al-Ani (2012) [25] used MOPSO algorithm to solve optimization problem of optimal pump operation for water distribution system using EPANET toolkit. Here they worked on bi-objective pump scheduling problem where objectives are: minimize the electrical energy cost and minimize the maintenance costs in terms of the total number of pump switches. In additional to the bi-objective pump-operational problem, pressure and tank levels (i.e., initial, minimum, and maximum) are considered as constraints in that paper for computing the most cost-effective solutions. The results showed that MOPSO produced most economical solutions.

Elizabeth F. G. Goldbarg (2006) [26] successfully used MOPSO for solving bi – objective degree constrained minimum spanning tree. They used operators for particle’s velocity based on local search and path-relinking procedures. And also, they showed the effectiveness of their solution by comparing it with evolutionary algorithms for the same problem.

M. Janga Reddy and D. Nagesh Kumar (2007) [27] used MOPSO based on pareto dominance for generating optimal trade-offs in reservoir operation. Additionally, they introduced variable size external repository and efficient elitist-mutation(EM) operator as modifications. And this variance is called EM-MOPSO due to additional mutation operator. They even showed that this approach is simple and effective enough to consider as viable alternative to solve multi-objective water resource and hydrology problems.

2.6 MOPSO -GT (Multi – Objective PSO with GT)

Zhiyoun Li et al., 2009 published a paper titled [28] “Multi-Objective Particle Swarm Optimization Algorithm Based on Game Strategies” which was combining MOPSO with concepts of game theory in order to solve multi- objective problems. This approach was inspired by Game Strategies, where objectives are considered as independent agent which is trying to optimize its own function. So multi – player game model was adopted into multi – objective PSO. So, performance was depended on appropriate game strategies. Here they used a bargain strategy for that purpose. Further they tested this algorithm with several benchmark functions and showed that its validity.

Pareto based quantum PSO method was used for this algorithm as MOPSO and external archive for storing solutions also used and weighted inertia equation was used as velocity update mechanism.

C.K. Goh (2010) et.al [29] **CCPSO** used some of the game theory ideas although they did not specially highlight it for solving multi – objective optimization problems using Game Theory. The main idea was to add competitive and cooperative co-evolutionary ideas to MOPSO. Canonical co-evolutionary paradigm can be broadly classified into two main categories

1. competitive co-evolution - the various subpopulations will always fight to
gain an advantage over the others.

2. cooperative co-evolution - subpopulations will exchange information
within each other during the evolutionary process

In their approach, they decomposed the problem in the search space and decision variables were evolved by different species, or subswarms. And assignment of decision variables to different subswarm was adapted by the competitive mechanism. Instead of ideal scenario where all particles from the subswarm compete with all other particles from other subswarm here they assigned a probability for each swarm for representing particular variable and only two subswarms, the current and competitor swarm was able to compete for right to represent any variable at a given time. And also, they utilized weighted inertia PSO based MOPSO method which had archiving technique as their base model. Based on their results they showed that this approach's effectiveness and validity

Kiran K. Annamdas (2009) [30] took an approach to solve multi – objective problem by combining modified game theory (MGT) with MOPSO and utilized closet discrete approach for solving problems with discrete design variables. And also, they illustrated its capabilities by applying it for solving several engineering problems and comparing its results with existing methods. In the proposed modified (cooperative) game theory approach, all the players are assumed to agree to find a compromise solution according to a mutually agreeable bargaining model or super-criterion. Modified Game Theory approach was presented by Rao and Freiheit (1991).

2.7 Hybrid MOPSO methods

Here we are focusing on some of the other methods which we are not discussed above but has an interesting insight in to MOO.

V Jančauskas - 2016 [31] introduced two new MOPSO methods based on notion of heterogeneous PSO - having several different types of particle in the same swarm. They all share information via same external pareto archive. The main idea is to use different particles to achieve different performance aspects of the MOPSO so performance will result them counter-acting each weakness. For example, one particle might focus on getting close to pareto front and other is trying to covering pareto front uniformly as possible so as a result they end up achieving both goals.

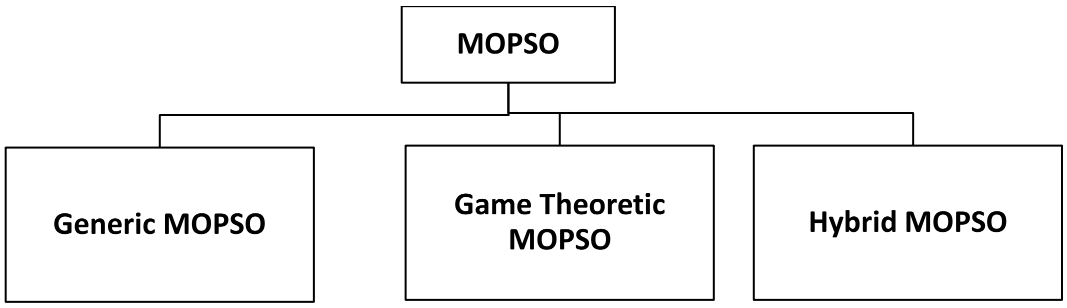
They proposed two algorithms first one ***HMOPSO-1*** has two different particle types called “sigma-particle” and “spread-particle”, here sigma particles are focused on achieving pareto front and spread-particles used to unsure uniform distribution among pareto front. For the second method ***HMOPSO-2*** they used three particle types “sigma-particle”, “spread-particle” and “closest-particle” here closest-particle works the same way as sigma-particle with exception which it considers closet in terms of Euclidean distance in objective space. Using those method, they showed the superiority of their method by test and comparing them against other several existing methods.

W.R.M.U.K. Wickramasinghe and X. Li - 2008 [32]

In this research they have incorporate differential-evolution method in to MOPSO. Differential-evolution itself is a method used in evolutionary computing for optimization. It optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It does not use gradient of the problem being optimized which means it does not want problem to be differentiable as is required by classic optimization methods and also can optimize problems which are not continuous or noisy. In this research authors proposed a new leader selection mechanism which is based on Differential-evolution which can successfully guide other particles towards pareto optimal front. And also, they have shown that considering three particle using DE is better experimentally and this combined MOPSO method is better model for optimization by testing it against several test functions and comparing it with other optimization methods.

2.8 A taxonomy of existing MOPSO approaches

It can be categorized into the following three branches



- Handling MO with PSO
  - Carlos A. Coello Coello et al. [1]
- Particle Swarm Optimizers for Pareto Optimization with Enhanced Archiving Techniques - Thomas Bartz Beielstein et al. [35]
- Particle Swarm Optimization for the Bi-objective Degree Constrained Minimum Spanning Tree - Elizabeth F. G. Goldberg e al., 2006 [26]
- Multi-objective optimization using dynamic neighborhood PSO - Xiaohui Hu I, & Russell Eberhart, 2002 [18]
- Multi-objective particle swarm optimization for generating optimal trade-offs in reservoir operation - M. Janga Reddy & D. Nagesh Kumar,2007 [27]
- A Modified PSO Algorithm for Constrained Multi-Objective Optimization
  - Lily D Li et al., 2009 [36]

- Multi-objective optimization of engineering systems using game theory and particle swarm optimization
  - Kiran K. Annamdas a & Singiresu S. Rao, 2013 [30]
- Multi-Objective Particle Swarm Optimization Algorithm Based on Game Strategies
  - Zhiyoun Li et al., 2009 [28]
- Handling multi-objective optimization problems with a multi-swarm cooperative particle swarm optimizer
  - Yong Zhang et al., 2011 [29]

- Evaluating the Performance of Multi-Objective Particle Swarm Optimization Algorithms
  -V Jančauskas - 2016 [31]
- Choosing Leaders for Multi-objective PSO Algorithms Using Differential Evolution
  -W.R.M.U.K. Wickramasinghe and X. Li - 2008 [32]

Figure 1. Taxonomy of MPSO approaches

2.9 Conclusions

In this chapter, starting from formal introduction of standard PSO we have discussed its extensions for solving MO problems as well as their various application researches conducted based on this MOPSO method. Then next we presented several successful attempts which some researchers had come up with combining game theory as well as evolutionary game theory in to MOO. And last section we further presented some interesting ideas which defining different particle behavior and types in to MOPSO concept. Based on those, it is clear that approaching MOO problems by considering particle behavior is an interesting area. For that we can use game theory as well as add the idea of competition which is not used in standard method, into cooperate behavior based MOPSO method and explore the possibilities.

Chapter 3 - Proposed method

In this chapter, new game theory based MOPSO method is purposed, namely "personality changing MOPSO". This method is also based on pareto dominance concept and utilize an external archiving mechanism to leader selection process.

3.1 "Personality changing MOPSO"

The essence of the proposed method is that there is only one particle type exist in the swarm and for each particle it has two personalities. One personality is trying to maximize/minimize its own outcome (one of the objective function) and other personality is trying to do the same for its own (other objective function). So, the notion of competition is modeled. Change of the personality is model in turn-based manner, initially randomly selected personality given the initial turn. For a given arbitrary moment (iteration) if one personality got its turn to decide, it tries to maximize its outcome but without degrading others. So previous information is available and used for this, if value suggested through position updating equation of the PSO is a better solution than previous one then it takes its chance to move to that position and if not, it gives its turn to other personality and so on. And the mean time if particle found non-dominated solution then it stores it in the archive or update it with later solution which dominates the existing one and these values were used to guide the flight of particles towards pareto optimal solutions. In this manner until the termination condition is met this process is repeated. After that particles themselves are moved towards pareto optimal solutions. Also, mutation operator is used to re-spawn weak solutions in to more promising area and this effect is decreased as iteration goes. So finally, we can get particles cost values as solutions for the MOO problem we considered.

Here we used one of the main branch of the game theory called "Sequential Game Theory" for decision making. Basic idea is to use previous information to decide current situation. The standard and conventional MOPSO model starts with particles with randomly assigned positions and velocities and then with time they slowly guide their flight, but particles going through obviously bad solutions as well, but only the repository updating mechanism has taken care of the collecting good solutions. But in our purposed method particles guide their flight in logical manner without going for obviously bad solutions. And the key idea behind this mechanism is the competition within personalities. So different from standard MOPSO which is based on cooperate behavior among particles in the swarm and our method additionally utilizes the competitive behavior as well.

3.2 Main Algorithm

Here personality 1 is trying to minimize first objective function and personality 2 is trying to minimize second objective and two-objective minimization is considered.
Up to 7th step initialization phase is given and then enter the while loop with given number of iterations and inside the while loop game theoretic decision-making process is taken place.

```
1. Initialize the population POP:
    a. For i=0 to MAX /* MAX = number of Particles */
    b. Initialize POP[i]
    c. Assign random personality /* 1 or 2 */
2. Initialize the speed of each particle:
    a. For i=0 to MAX
    b. VEL[i] = 0
3. Evaluate each of the particle in POP (Calculate & assign cost values for given two-objective
   functions).
4. Store the positions of the particles that represent nondominated vectors in the repository
   REP (for leader selection process)
5. Generate hypercubes of the search space explored so far, and locate the particles using
   these hypercubes as a coordinate system where each particle's coordinates are defined
   according to the values of its objective functions.
6. Initialize the memory of each particle
    a. For i=0 to MAX
    b. PBEST[i] = POP[i]
7. WHILE maximum number of cycles has not been reached DO
    a. Compute the speed of each particle using the following equation:
        VEL[i] = W*VEL[i] + $R_1$*(PBEST[i] - POP[i]) + $R_2$*(REP[h] - POP[i])
        Where W is the inertia weight and $R_1$, $R_2$ are random numbers in the range [0...1].
        PBEST[i] is the best position that the particle i has found REP[h] is a value taken
        from the repository. POP[i] is the current value of the particle i.
    b.  Compute the new position of the particle.
        POP[i] = POP[i] + VEL[i]
    c.  Maintain particles within the search space boundary
    d.  Evaluate each particle in POP:
        Calculate new cost values for the particle i and compare it with old cost values, if
        personality is type 1 and if it is possible to minimize its cost value by taking new
        value without degrading other cost value, take new value and stay in same
        Personality. Else change personality and give others the chance. /* personality
        type 2 will also follows same reasoning */
    e.  Apply mutation for re-positioning bad values.
    f.  Update the content of REP and eliminate dominated locations.
        /* This repository is use for leader selection and guiding the flight of the particles*/
    g.  Increment the loop counter
8.  END WHILE
```

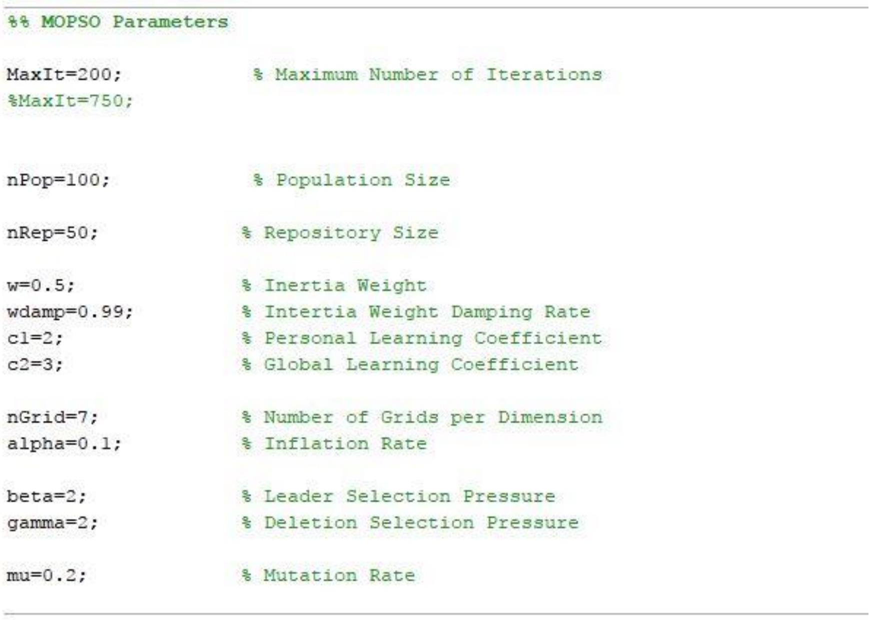
Chapter 4 - Implementation

4.1 Introduction

In this chapter implantation of the PC-MOPSO is given and all Implementation was done using MATLAB 2015a version.

4.1.1 Parameter definition

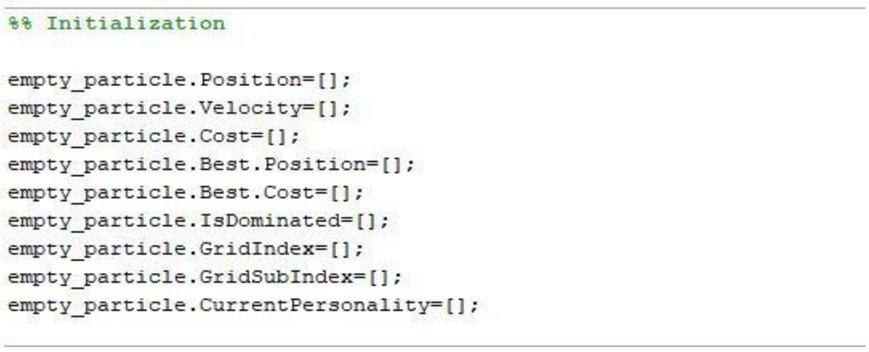
The following set of parameters were set, and adjusted for different scenarios.



```
%% MOPSO Parameters

MaxIt=200;           % Maximum Number of Iterations
%MaxIt=750;


nPop=100;            % Population Size

nRep=50;            % Repository Size

w=0.5;              % Inertia Weight
wdamp=0.99;         % Intertia Weight Damping Rate
c1=2;               % Personal Learning Coefficient
c2=3;               % Global Learning Coefficient

nGrid=7;            % Number of Grids per Dimension
alpha=0.1;          % Inflation Rate

beta=2;             % Leader Selection Pressure
gamma=2;            % Deletion Selection Pressure

mu=0.2;             % Mutation Rate
```

4.1.2 Particle Initialization

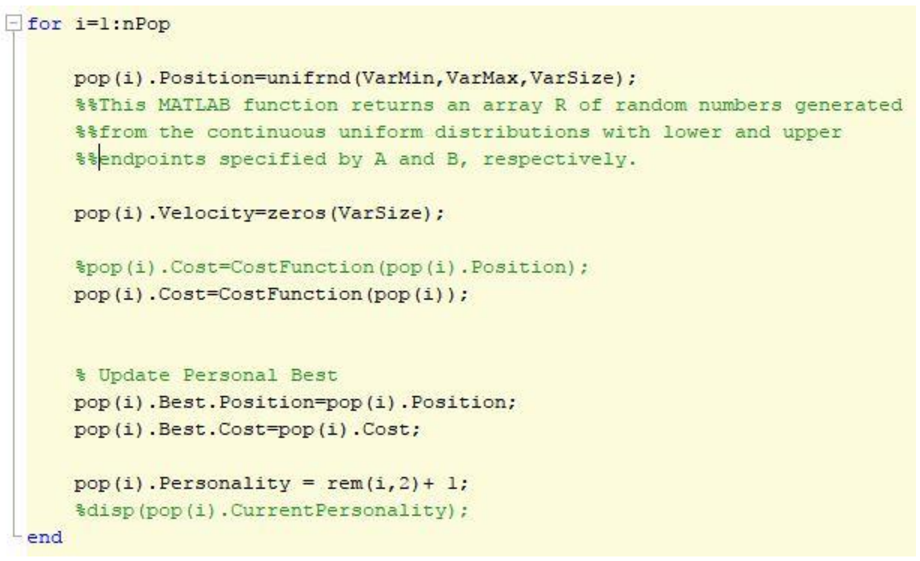
Particle initial settings are given below.



```
%% Initialization

empty_particle.Position=[];
empty_particle.Velocity=[];
empty_particle.Cost=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];
empty_particle.IsDominated=[];
empty_particle.GridIndex=[];
empty_particle.GridSubIndex=[];
empty_particle.CurrentPersonality=[];
```

4.1.3 Calculate initial values for position, velocity and personality and Cost

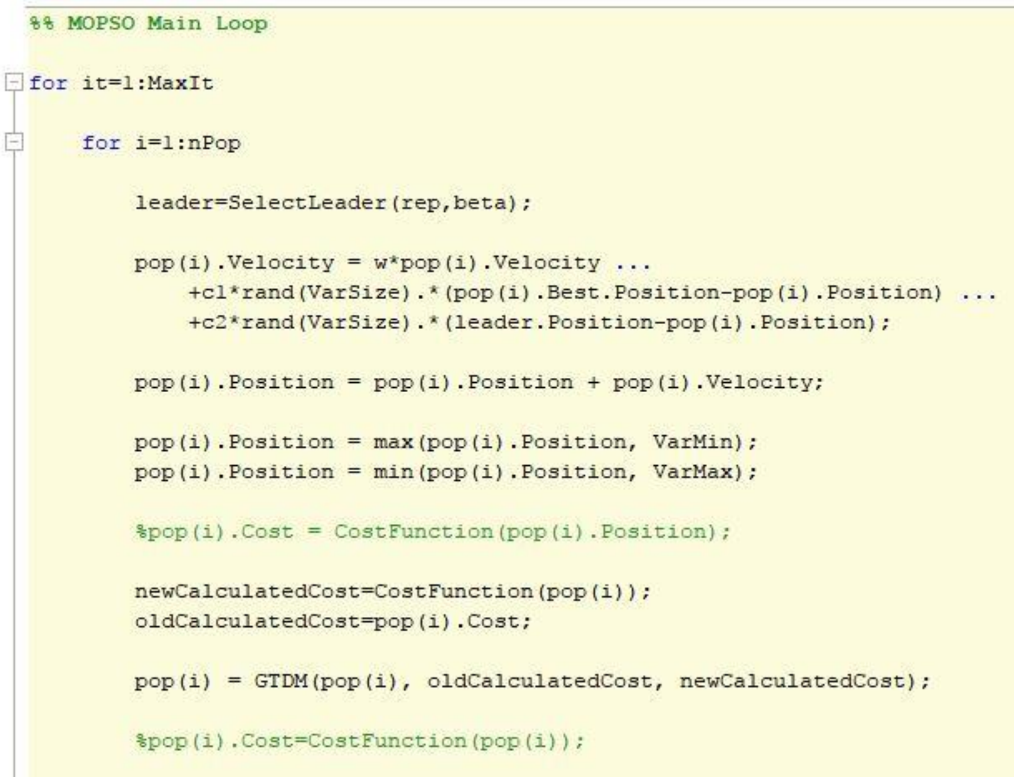
Evaluate each of the particle in POP /*Calculate & assign cost values for given two-objective functions and assign personality type*/.



```
for i=1:nPop

    pop(i).Position=unifrnd(VarMin,VarMax,VarSize);
    %%This MATLAB function returns an array R of random numbers generated
    %%from the continuous uniform distributions with lower and upper
    %%endpoints specified by A and B, respectively.

    pop(i).Velocity=zeros(VarSize);

    %pop(i).Cost=CostFunction(pop(i).Position);
    pop(i).Cost=CostFunction(pop(i));


    % Update Personal Best
    pop(i).Best.Position=pop(i).Position;
    pop(i).Best.Cost=pop(i).Cost;

    pop(i).Personality = rem(i,2)+ 1;
    %disp(pop(i).CurrentPersonality);
end
```

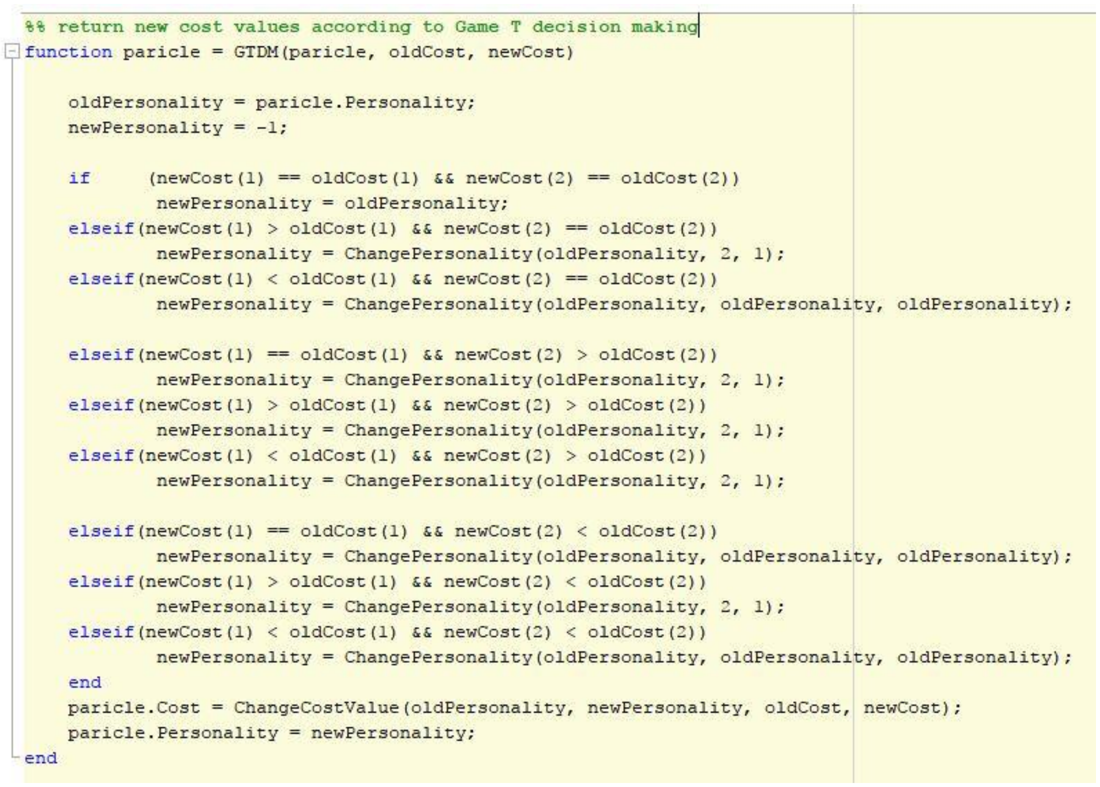
4.1.4 Main Loop

following is the main loop of the algorithm for PC-MOPSO, in each iteration leader selection, position and velocity updating, damping the velocity and positions and game theoretic decision-making process is taken place.



```
%% MOPSO Main Loop

for it=1:MaxIt

    for i=1:nPop

        leader=SelectLeader(rep,beta);

        pop(i).Velocity = w*pop(i).Velocity ...
            +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position) ...
            +c2*rand(VarSize).*(leader.Position-pop(i).Position);

        pop(i).Position = pop(i).Position + pop(i).Velocity;

        pop(i).Position = max(pop(i).Position, VarMin);
        pop(i).Position = min(pop(i).Position, VarMax);

        %pop(i).Cost = CostFunction(pop(i).Position);

        newCalculatedCost=CostFunction(pop(i));
        oldCalculatedCost=pop(i).Cost;

        pop(i) = GTDM(pop(i), oldCalculatedCost, newCalculatedCost);

        %pop(i).Cost=CostFunction(pop(i));
```

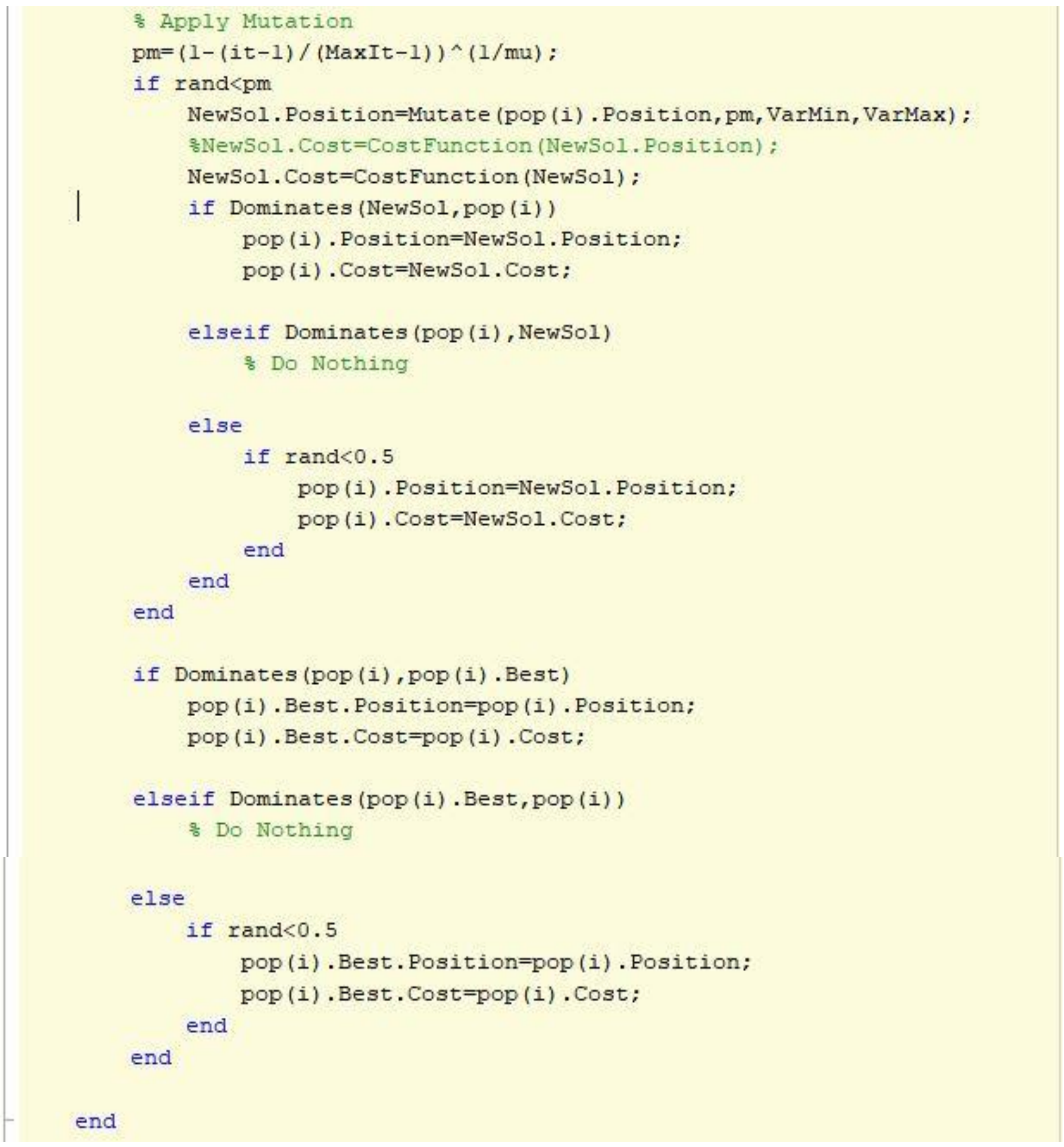
The GTDM function is defined as follow for Game theoretic decision-making process, this function takes old cost value, new cost value and particle as input and output the modified particle., if personality is type 1 and if it is possible to minimize its cost value by taking new value without degrading other cost value, take new value and stay in same Personality. Else change personality and give others the chance. Same reasoning is used by personality type 2.



```
%% return new cost values according to Game T decision making
function paricle = GTDM(paricle, oldCost, newCost)

    oldPersonality = paricle.Personality;
    newPersonality = -1;

    if      (newCost(1) == oldCost(1) && newCost(2) == oldCost(2))
             newPersonality = oldPersonality;
    elseif(newCost(1) > oldCost(1) && newCost(2) == oldCost(2))
             newPersonality = ChangePersonality(oldPersonality, 2, 1);
    elseif(newCost(1) < oldCost(1) && newCost(2) == oldCost(2))
             newPersonality = ChangePersonality(oldPersonality, oldPersonality, oldPersonality);

    elseif(newCost(1) == oldCost(1) && newCost(2) > oldCost(2))
             newPersonality = ChangePersonality(oldPersonality, 2, 1);
    elseif(newCost(1) > oldCost(1) && newCost(2) > oldCost(2))
             newPersonality = ChangePersonality(oldPersonality, 2, 1);
    elseif(newCost(1) < oldCost(1) && newCost(2) > oldCost(2))
             newPersonality = ChangePersonality(oldPersonality, 2, 1);

    elseif(newCost(1) == oldCost(1) && newCost(2) < oldCost(2))
             newPersonality = ChangePersonality(oldPersonality, oldPersonality, oldPersonality);
    elseif(newCost(1) > oldCost(1) && newCost(2) < oldCost(2))
             newPersonality = ChangePersonality(oldPersonality, 2, 1);
    elseif(newCost(1) < oldCost(1) && newCost(2) < oldCost(2))
             newPersonality = ChangePersonality(oldPersonality, oldPersonality, oldPersonality);
    end
    paricle.Cost = ChangeCostValue(oldPersonality, newPersonality, oldCost, newCost);
    paricle.Personality = newPersonality;
end
```

4.1.5 Apply mutation

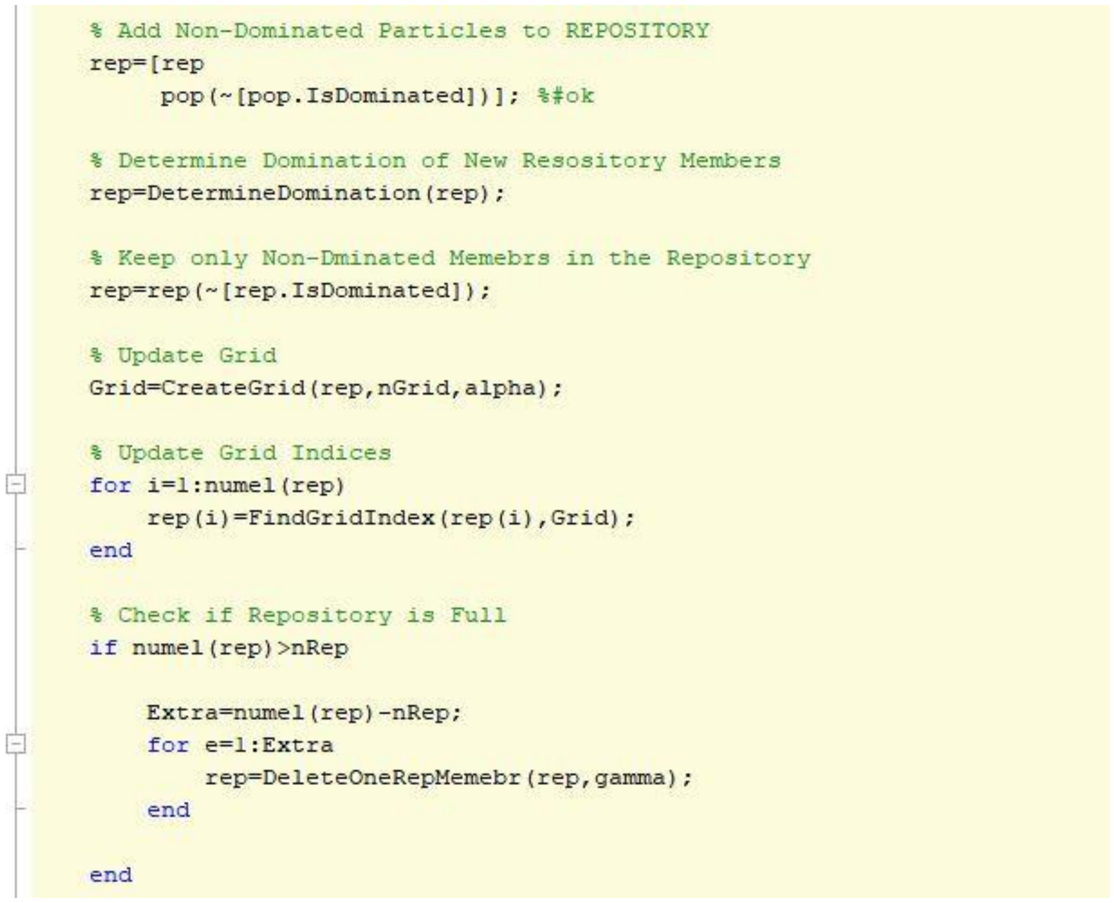
The mutation operator used here is the same mutation operator used in standard -MOPSO.



```
        % Apply Mutation
        pm=(1-(it-1)/(MaxIt-1))^(1/mu);
        if rand<pm
            NewSol.Position=Mutate(pop(i).Position,pm,VarMin,VarMax);
            %NewSol.Cost=CostFunction(NewSol.Position);
            NewSol.Cost=CostFunction(NewSol);
            if Dominates(NewSol,pop(i))
                pop(i).Position=NewSol.Position;
                pop(i).Cost=NewSol.Cost;

            elseif Dominates(pop(i),NewSol)
                % Do Nothing

            else
                if rand<0.5
                    pop(i).Position=NewSol.Position;
                    pop(i).Cost=NewSol.Cost;
                end
            end
        end

        if Dominates(pop(i),pop(i).Best)
            pop(i).Best.Position=pop(i).Position;
            pop(i).Best.Cost=pop(i).Cost;

        elseif Dominates(pop(i).Best,pop(i))
            % Do Nothing

        else
            if rand<0.5
                pop(i).Best.Position=pop(i).Position;
                pop(i).Best.Cost=pop(i).Cost;
            end
        end

    end
```

4.1.6 Updating Repository

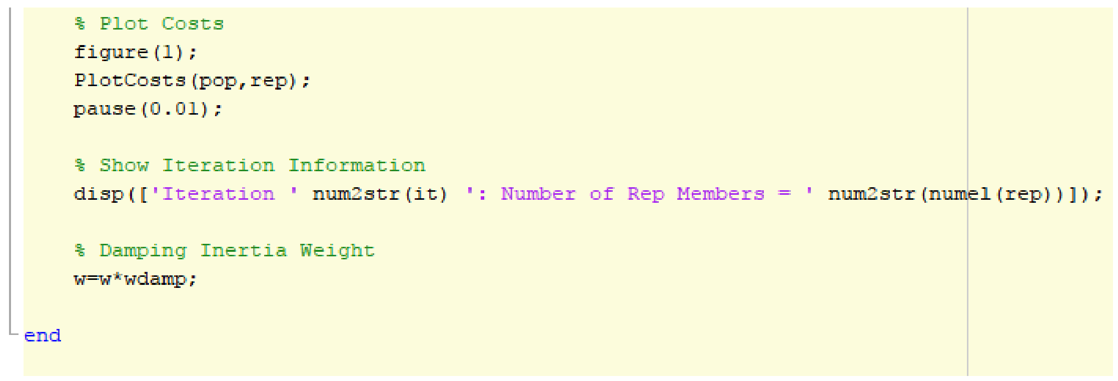
The following code Update the content of repository and eliminate dominated locations and then update the grid.



```
% Add Non-Dominated Particles to REPOSITORY
rep=[rep
     pop(~[pop.IsDominated])]; %#ok

% Determine Domination of New Resository Members
rep=DetermineDomination(rep);

% Keep only Non-Dminated Memebrs in the Repository
rep=rep(~[rep.IsDominated]);

% Update Grid
Grid=CreateGrid(rep,nGrid,alpha);

% Update Grid Indices
for i=1:numel(rep)
    rep(i)=FindGridIndex(rep(i),Grid);
end

% Check if Repository is Full
if numel(rep)>nRep

    Extra=numel(rep)-nRep;
    for e=1:Extra
        rep=DeleteOneRepMemebr(rep,gamma);
    end

end
```

4.1.7 Plot the results

Here in each iteration plotting the particle movement in the objective space is done. By using this we can visually see what is going on clearly.



```
    % Plot Costs
    figure(1);
    PlotCosts(pop,rep);
    pause(0.01);

    % Show Iteration Information
    disp(['Iteration ' num2str(it) ': Number of Rep Members = ' num2str(numel(rep))]);

    % Damping Inertia Weight
    w=w*wdamp;

end
```

Chapter 5 - Results and Evaluation

5.1 Introduction

To test and compare the proposed method, several standard test functions were used, it mainly compared with standard MOPSO (Coello Coello) [1] and available generated true pareto sets for the test functions. And well-established metric for MOO, hypervolume indicator is used to measure quality of the pareto fronts. And it measures the volume of the dominated portion of the objective space. We have used existing implementation of hypervolume indicator approximation for this purpose.

Testing was done using intel core i3 with 4GB ram computer with Windows 10 operating system. And for the implementation MatLab 2015a version was used.

5.2 Test function 1 (ZDT1) [33]

This is a function of n variables given by following mathematical equations. And simultaneously minimizing of f1 and f2 functions are considered.

Formulation:

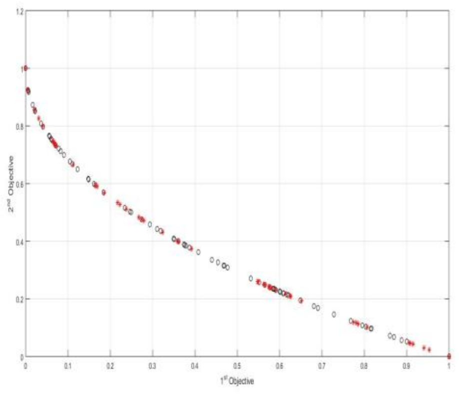
$$f_1 = x_1$$

$$f_2 = g \cdot \left(1.0 - \sqrt{\frac{f_1}{g}}\right)$$

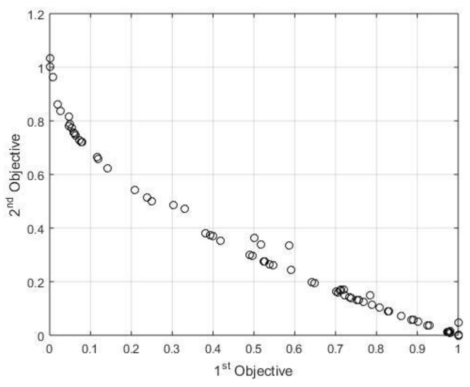
$$g(x_2, \ldots, x_n) = 1.0 + \frac{9}{n-1} \sum_{i=2}^{n} x_i$$

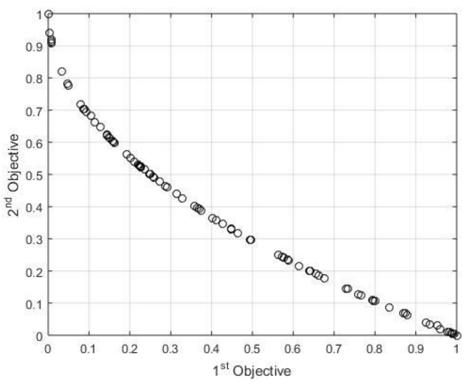
$$0 \leq x_i \leq 1, i = 1, \ldots, n$$

We compare one standard MOPSO & proposed PC-MOPSO for different number of iterations.



(a)





(b) *(c)*

Figure 2: Comparison of output graphs in objective space for (*a) standard MOPSO for ZDT1 200 iterations, (b) PC-MOPSO for ZDT1 200 iterations, (c) PC-MOPSO for ZDT1 750 iterations*

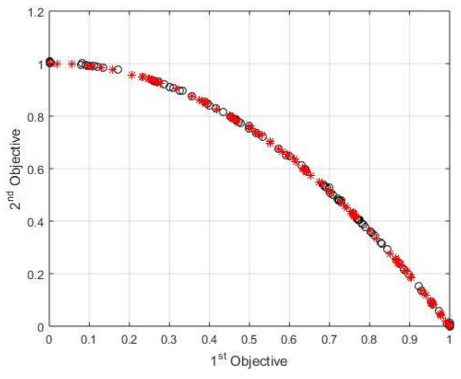
5.3 Test function 2 (ZDT2) [33]

This is a function of n variable given by following mathematical equations. And simultaneously minimizing of f1 and f2 functions are considered.

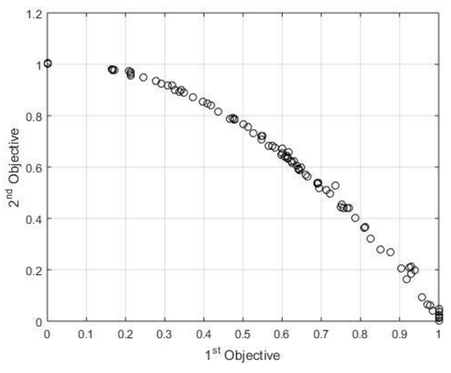
Formulation:

$$f_1 = x_1$$
$$f_2 = g(\vec{x}) \cdot \left[1.0 - (x_1/g(\vec{x}))^2\right]$$
$$g(\vec{x}) = 1 + \frac{9}{n-1}\left(\sum_{i=2}^{n} x_i\right)$$
$$0 \le x_i \le 1, i = 1, \ldots, n$$

We compare MOPSO with PC-MOPSO for 200 iterations. In red-color (*) solutions of MOPSO is presented.



(a)



(b)

***Figure 3: Comparison of output graphs in objective space for** (a) standard MOPSO for ZDT2 200 iterations, (b) PC-MOPSO for ZDT1 200 iterations*

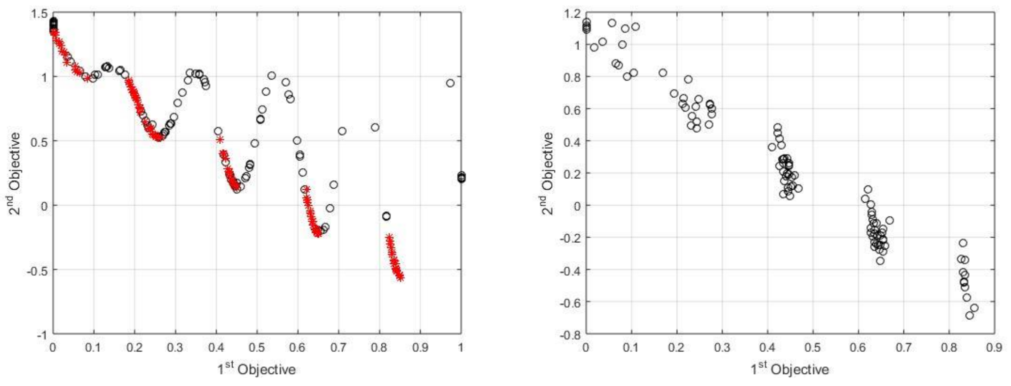
5.4 Test function 3 (ZDT3) [33]

This function is given by following mathematical equations and multiple fronts of pareto solutions are expected.

Formulation:

$$f_1(\vec{x}) = x_1$$
$$f_2(\vec{x}) = g(\vec{x})[1 - \sqrt{x_1/g(\vec{x})} - x_1/g(\vec{x})\sin(10\pi x_1)$$
$$g(\vec{x}) = 1 + \frac{9}{n-1}\left(\sum_{i=2}^{n} x_i\right)$$
$$0 \le x_i \le 1,\ i = 1, \ldots, n$$

Comparison of MOPSO with PC-MOPSO for 200 iterations are given by following graphs.



(a) *(b)*

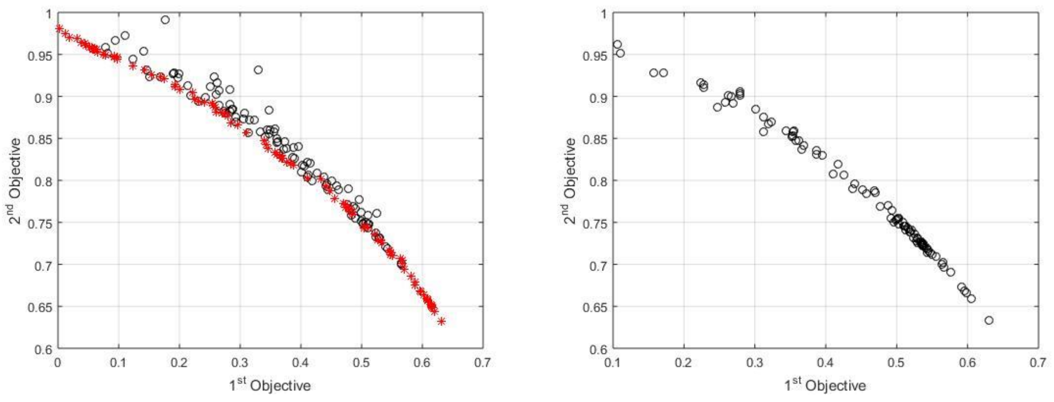
***Figure 4**: **Comparison of output graphs in objective space for** (a) standard MOPSO for ZDT3 200 iterations, (b) PC-MOPSO for ZDT3 200 iterations*

5.5 Test function 4 - Fonseca and Fleming function [34]

In following mathematical equations, simultaneous minimization of f1 and f2 functions considered.

$$\text{Minimize} = \begin{cases} f_1(\boldsymbol{x}) & = 1 - \exp\left[-\sum_{i=1}^{n}\left(x_i - \frac{1}{\sqrt{n}}\right)^2\right] \\ f_2(\boldsymbol{x}) & = 1 - \exp\left[-\sum_{i=1}^{n}\left(x_i + \frac{1}{\sqrt{n}}\right)^2\right] \end{cases}$$

Comparison of MOPSO with PC-MOPSO for 200 iterations are given by following graphs.



(a) (b)

Figure 5: Comparison of output graphs in objective space for *(a) standard MOPSO 200 iterations, (b) PC-MOPSO 200 iterations for Fonseca and Fleming function*

Table 5-1: *Parameter setting used for testing in PC-MOPSO*

	Population Size	**repository size**	**C1(in velocity equation)**	**C2(in velocity equation)**	**Mutation rate**
ZDT1, ZDT3, FF	100	50	1	2	0.1
ZDT2	100	50	2	3	0.25

Table 5-2: *Approximaed hypervolume values for each test functions.*

	ZDT1	**ZDT2**	**ZDT3**	**Fonseca and Fleming function**
Standard MOPSO	0.6396(96.47%)	0.3198(97.17%)	0.8335(0.069% ovr)	0.0867
PC-MOPSO	**0.6583(99.26%)**	0.3162(96.08%)	0.7612(97.62%)	0.0748
Available results	0.6630	0.3291	0.7797	

Table 5-3: *Change of hypervolume values with iterations in PC-MOPSO.*

	200 iterations	**300 iterations**	**500 iterations**
ZDT1	0.6405	0.6491	0.6551
ZDT2	0.3005	0.3082	0.3232
ZDT3	0.9064	0.8342	0.8034

According to the approximated hypervolume values (higher is considered as better) it was shown that for ZDT1 type functions which has convex pareto optimal front, PC-MOPSO *(Table 5-2, first column)* perform slightly better with increasing iterations. And for the ZDT2 which has non-convex pareto front it performs similar to other method (*Table 5-2, second column)*. But we have to adjust parameters, it was shown that in other parameter setting it was not able to converge to the solution front. But for the ZDT3 type functions which has multiple convex pareto front its performance is not very good (*Table 5-2, third column)*. This can be understood since in this method each particle is trying reach pareto front and weak solutions were re-positioned using mutation operator so particles often confused on which front to reach this may leads to converging into few pareto fronts. And with increasing number of iterations it was shown that solution became better.

Overall, we can say that proposed method performs good at handing ZDT1 and ZDT2 type problems and with increasing number of iterations quality of the solutions increased but further research is needed on enhancing its results on ZDT3 type problems.

Other interesting thing which is unique to PC-MOPSO is unlike in other MOPSO methods which used repository to store solutions and used those final repository values as final solutions to the problem, PC-MOPSO particle itself try to reach pareto front. so, in other words it behaves similar to evolutionary algorithm/genetic algorithms by incorporating notion of competition and GT decision making.

Chapter 6 - Conclusions

6.1 introduction

In this research we primarily focused on behavioral aspect of the PSO method and effects on using game theory-based concepts such as competition which is not used in conventional methods to achieve MOO. For that we came up with new method which is based on each particle has two personalities and each personality act rationally to maximize its own benefit. In other words, we have embedded notion of competition using game theory in abstract manner. And in Experiments, it was shown that proposed method is competitive and capable of handling most of the problems which we considered, although further research may be needed for certain areas.

Although general PSO method is based on natural phenomena o bird flocking or fish schooling, when come to MOPSO although it uses same mechanism to fly, it utilizes other mechanism which seemingly unnatural to achieve the final result. use of external archiving mechanism to store solutions is one of them. So, in a way we can say that in MOPSO final solutions are achieved through the side effects which has nothing to do with naturalist point of view. It is understandable since this is only “nature inspired “algorithms.

But in our purposed method (PC-MOPSO) each particle/personality act in rational manner and take decision in order to maximize its own benefits. This is more approximate to natural behavior and the most interesting thing is, as a result of this, particle itself is moving toward the solution. This kind of behavior can be observed in evolutionary and genetic algorithms which have underlying mechanism is based on evolutionary game theory where evolutionary stable strategy survived. So, by introducing game theory and notion of competition we can see that interesting behavior are emerged. I think this may lead in to an interesting research area.

6.2 Conclusions about research problem/hypothesis

As hypothesized in the research problem, by considering experiment results we can conclude that game theory based MOPSO method is appealing for multi-objective optimization (2.79% accuracy than MOPSO for ZDT1). And moreover, it leads to some interesting behavior. With further refinement and research this may open a door to very interesting branch of research in particle swarm metaphor.

6.3 Limitations

In this research we only considered optimizing two objective functions. And also, we have omitted constrained optimization as well. This method did not perform very well for problems which has multiple pareto fronts. and also, value of the mutation rate had to be fine tuned in some problems to achieve best results.

6.4 Implications for further research

- Refinement of this method for handling multiple pareto front and achieving diversity
- Investigating similarity between this method (PC-MOPSO) behavior with evolutionary algorithms.
- Effects of using different mutation operators / leader selection mechanism
- Use of this method for constrained and many objective optimizations.

References

[1] C. A.C. Coello,G. T. Pulido,M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation,* vol. 8, no. 3, pp. 256-279 , 2004.

[2] Kennedy, J. and Eberhart, R, "Particle Swarm Optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, 1995.

[3] Jun Sun ,Wei Fang ,Xiaojun Wu,Vasile Palade,Wenbo Xu, "Quantum-behaved particle swarm optimization," *Evolutionary Computation,* vol. 20, no. 3, pp. 349-393 , Fall 2012.

[4] Hesham Ahmed Hefny & Shahira Shaaban Azab, "Chaotic particle swarm optimization," in *Informatics and Systems (INFOS)*, 2010 .

[5] Dong-ping Tian & Nai-qian Li, "Fuzzy Particle Swarm Optimization Algorithm," in *Artificial Intelligence*, Hainan Island, China, 2009.

[6] Ali B. HashemiM. R. Meybodi, "Cellular PSO: A PSO for Dynamic Environments," in *International Symposium on Intelligence Computation and Applications*, 2009.

[7] Paul S. Andrews, Jon Timmis, "An Introduction to Artificial Immune Systems," in *Handbook of Natural Computing*, Springer Berlin Heidelberg, 2012, pp. 1575-1597.

[8] Fred Glover ,Manuel Laguna, Tabu Search, Kluwer Academic Publishers Norwell, MA, USA ©1997, 1997.

[9] C.A. Coello Coello, M.S. Lechuga, "MOPSO: a proposal for multiple objective particle swarm optimization," in *Evolutionary Computation*, 2002.

[10] J. C. Bansal, P. K. Singh, Mukesh Saraswat, "Inertia Weight strategies in Particle Swarm Optimization," in *Nature and Biologically Inspired Computing (NaBIC)*, 2011.

[11] J. Knowles ,D. Corne, "The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation," in *Evolutionary Computation*, Washington, 1999.

[12] K. Deb, A. Pratap,S. Agarwal, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation ,* vol. 6, no. 2, pp. 182 - 197, 2002.

[13] Margarita Reyes-Sierra , Carlos A. Coello Coello, "Multi-Objective Particle Swarm Optimizers:A Survey of the State-of-the-Art," *International Journal of Computational Intelligence Research.,* vol. 2, no. 3, p. 287–308, 2006.

[14] Kalyanmoy Deb, David E. Goldberg , "An Investigation of Niche and Species Formation in Genetic Function Optimization," *Proceedings of the 3rd International Conference on Genetic Algorithms,* pp. 42-50, 1989.

[15] David E. Goldberg ,Jon Richardson, "Genetic algorithms with sharing for multimodal function optimization," *International Conference on Genetic Algorithms,* pp. 41-49, 1987.

[16] Konstantinos E. Parsopoulos,Michael N. Vrahatis, "Genetic algorithms with sharing for multimodal function optimization," *ACM Symposium on Applied Computing ,* p. 603– 607, 2002.

[17] U. Baumgartner, Ch. Magele, W. Renhart, "Pareto optimality and particle swarm optimization," *IEEE Transactions on Magnetics,* vol. 40, no. 2, pp. 1172 - 1175, 2004.

[18] Xiaohui Hu ,Russell Eberhart, "Multiobjective optimization using dynamic neighborhood particle swarm optimization," *Evolutionary Computation,* vol. 2, p. 1677–1681, 2002.

[19] Chi-kin Chow , Hung-tat Tsui, "Autonomous agent response learning by a multi-species particle swarm optimization," *Evolutionary Computation ,* vol. 1, p. 778–785, 2004.

[20] Konstantinos E. Parsopoulos, Dimitris K. Tasoulis, and Michael N. Vrahatis, "Multiobjective optimization using parallel vector evaluated particle swarm optimization," *International Conference on Artificial Intelligence and Applications,* vol. 2, p. 823–828, 2004.

[21] Tapabrata Ray ,K.M. Liew, "A swarm metaphor for multiobjective design optimization," *Engineering Optimization ,* vol. 34, no. 2, pp. 141-153 , 2002.

[22] Jonathan E. Fieldsend , Sameer Singh, "A multiobjective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence," *UK Workshop on Computational Intelligence,* pp. 34-44, 2002.

[23] Mahdi Mahfouf, Min-You Chen, and Derek Arturh Linkens, "Adaptive weighted particle swarm optimisation for multi-objective optimal design of alloy steels," in *In Parallel Problem*, Birmingham, UK, 2004, p. 762–771.

[24] Zhang Xiao-hua,Meng Hong-yun,Jiao Li-cheng, "Intelligent particle swarm optimization in multiobjective optimization," in *IEEE* , Edinburgh, Scotland, UK , 2005.

[25] Dhafar Al-Ani,Saeid Habibi, "Optimal pump operation for water distribution systems using a new multi-agent Particle Swarm Optimization technique with EPANET," in *Electrical & Computer Engineering (CCECE)*, Montreal, QC, Canada , 2012.

[26] E.F.G. Goldbarg,G.R. de Souza,M.C. Goldbarg, "Particle Swarm Optimization for the Bi-objective Degree constrained Minimum Spanning Tree," in *Evolutionary Computation*, Vancouver, BC, Canada , 2006.

[27] Reddy MJ, Kumar DN, "Multi-objective particle swarm optimization for generating optimal trade-offs in reservoir operation," in *Hydrological Processes*, 2007.

[28] Zhiyong Li,Songbing Liu,Degui Xiao,Kenli Li, "Multi-objective particle Swarm optimization algorithm based on game strategies," in *Genetic and Evolutionary Computation Conference*, Shanghai, China, 2009.

[29] Yong Zhang, Dun-wei Gong, Zhong-hai Ding, "Handling multi-objective optimization problems with a multi-swarm cooperative particle swarm optimizer," *Expert Systems with Applications,* vol. 38, no. 11, pp. 13933-13941, 2011.

[30] Kiran K. Annamdas , Singiresu S. Rao, "Multi-objective optimization of engineering systems using game theory and particle swarm optimization," *Engineering Optimization ,* vol. 41, no. 8, pp. 737-752 , 2008.

[31] V. Janˇcauskas, "EVALUATING THE PERFORMANCE OF MULTI-OBJECTIVE PARTICLE," VILNIUS UNIVERSITY, Lithuania, 2016.

[32] Upali Wickramasinghe,Xiaodong Li, "Choosing Leaders for Multi-objective PSO Algorithms Using Differential Evolution," in *Asia-Pacific Conference on Simulated Evolution and Learning*, Springer, Berlin, Heidelberg, 2008.

[33] Eckart Zitzler,Kalyanmoy Deb,Lothar Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evolutionary Computation,* vol. 8, no. 2, pp. 173-195 , 2000 .

[34] Carlos M. Fonseca , Peter J. Fleming, "An Overview of Evolutionary Algorithms in Multiobjective Optimization," *Evolutionary Computation,* vol. 3, no. 1, pp. 1 - 16, 1995.

[35] T. Bartz-Beielstein, P. Limbourg, J. Mehnen, K. Schmitt, K. Parsopoulos, M. Vrahatis, "Particle swarm optimizers for Pareto optimization with enhanced archiving techniques," in *Evolutionary Computation*, Canberra, ACT, Australia, Australia , 2003.

[36] Lily D. Li, Xinghuo Yu, Xiaodong Li, "A Modified PSO Algorithm for Constrained Multi-objective Optimization," in *Network and System Security*, Gold Coast, QLD, Australia , 2009.

Appendix C: Code Listings

// main code - mopso.m

```
clc;
clear;
close all;

%% Problem Definition

%CostFunction=@(x) ZDT3(x);      % Cost Function

CostFunction=@(x) MOP2(x);
%CostFunction=@(x) MOP5(x);
%CostFunction=@(x) MOP4(x);
%CostFunction=@(x) MOP_new4(x);

%nVar=5;             % Number of Decision Variables
nVar=30;

VarSize=[1 nVar];   % Size of Decision Variables Matrix

VarMin=0;          % Lower Bound of Variables
VarMax=1;          % Upper Bound of Variables


%% MOPSO Parameters

MaxIt=200;           % Maximum Number of Iterations
%MaxIt=750;


nPop=100;            % Population Size

nRep=50;            % Repository Size

w=0.5;              % Inertia Weight
wdamp=0.99;         % Intertia Weight Damping Rate
c1=2;               % Personal Learning Coefficient
c2=3;               % Global Learning Coefficient

nGrid=7;            % Number of Grids per Dimension
alpha=0.1;          % Inflation Rate

beta=2;             % Leader Selection Pressure
gamma=2;            % Deletion Selection Pressure

mu=0.2;             % Mutation Rate

%% Initialization

empty_particle.Position=[];
empty_particle.Velocity=[];
empty_particle.Cost=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];
empty_particle.IsDominated=[];
empty_particle.GridIndex=[];
empty_particle.GridSubIndex=[];
empty_particle.CurrentPersonality=[];

%% This MATLAB function returns an array containing n copies of A in the row
and column dimensions
pop=repmat(empty_particle,nPop,1);

for i=1:nPop

    pop(i).Position=unifrnd(VarMin,VarMax,VarSize);
    %%This MATLAB function returns an array R of random numbers generated
    %%from the continuous uniform distributions with lower and upper
    %%endpoints specified by A and B, respectively.

    pop(i).Velocity=zeros(VarSize);

    %pop(i).Cost=CostFunction(pop(i).Position);
    pop(i).Cost=CostFunction(pop(i));


    % Update Personal Best
    pop(i).Best.Position=pop(i).Position;
    pop(i).Best.Cost=pop(i).Cost;

    pop(i).Personality = rem(i,2)+ 1;
    %disp(pop(i).CurrentPersonality);
end

%return;

% Determine Domination
pop=DetermineDomination(pop);

rep=pop(~[pop.IsDominated]);

Grid=CreateGrid(rep,nGrid,alpha);

for i=1:numel(rep)
    rep(i)=FindGridIndex(rep(i),Grid);
end


%% MOPSO Main Loop

for it=1:MaxIt

    for i=1:nPop

        leader=SelectLeader(rep,beta);

        pop(i).Velocity = w*pop(i).Velocity ...
            +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position) ...
            +c2*rand(VarSize).*(leader.Position-pop(i).Position);
        pop(i).Position = pop(i).Position + pop(i).Velocity;

        pop(i).Position = max(pop(i).Position, VarMin);
        pop(i).Position = min(pop(i).Position, VarMax);

        %pop(i).Cost = CostFunction(pop(i).Position);

        newCalculatedCost=CostFunction(pop(i));
        oldCalculatedCost=pop(i).Cost;

        pop(i) = GTDM(pop(i), oldCalculatedCost, newCalculatedCost);

        %pop(i).Cost=CostFunction(pop(i));

        % Apply Mutation
        pm=(1-(it-1)/(MaxIt-1))^(1/mu);
        if rand<pm
            NewSol.Position=Mutate(pop(i).Position,pm,VarMin,VarMax);
            %NewSol.Cost=CostFunction(NewSol.Position);
            NewSol.Cost=CostFunction(NewSol);
            if Dominates(NewSol,pop(i))
                pop(i).Position=NewSol.Position;
                pop(i).Cost=NewSol.Cost;

            elseif Dominates(pop(i),NewSol)
                % Do Nothing

            else
                if rand<0.5
                    pop(i).Position=NewSol.Position;
                    pop(i).Cost=NewSol.Cost;
                end
            end
        end

        if Dominates(pop(i),pop(i).Best)
            pop(i).Best.Position=pop(i).Position;
            pop(i).Best.Cost=pop(i).Cost;

        elseif Dominates(pop(i).Best,pop(i))
            % Do Nothing

        else
            if rand<0.5
                pop(i).Best.Position=pop(i).Position;
                pop(i).Best.Cost=pop(i).Cost;
            end
        end

    end

    % Add Non-Dominated Particles to REPOSITORY
    rep=[rep
         pop(~[pop.IsDominated])]; %#ok

    % Determine Domination of New Resository Members
    rep=DetermineDomination(rep);
    % Keep only Non-Dminated Memebrs in the Repository
    rep=rep(~[rep.IsDominated]);

    % Update Grid
    Grid=CreateGrid(rep,nGrid,alpha);

    % Update Grid Indices
    for i=1:numel(rep)
        rep(i)=FindGridIndex(rep(i),Grid);
    end

    % Check if Repository is Full
    if numel(rep)>nRep

        Extra=numel(rep)-nRep;
        for e=1:Extra
            rep=DeleteOneRepMemebr(rep,gamma);
        end

    end

    % Plot Costs
    figure(1);
    PlotCosts(pop,rep);
    pause(0.01);

    % Show Iteration Information
    disp(['Iteration ' num2str(it) ': Number of Rep Members = '
num2str(numel(rep))]);

    % Damping Inertia Weight
    w=w*wdamp;

end

%% Resluts
```

// GTDM.m

```
%% return new cost values according to Game T decision making
function paricle = GTDM(paricle, oldCost, newCost)

    oldPersonality = paricle.Personality;
    newPersonality = -1;

    if     (newCost(1) == oldCost(1) && newCost(2) == oldCost(2))
            newPersonality = oldPersonality;
    elseif(newCost(1) > oldCost(1) && newCost(2) == oldCost(2))
            newPersonality = ChangePersonality(oldPersonality, 2, 1);
    elseif(newCost(1) < oldCost(1) && newCost(2) == oldCost(2))
            newPersonality = ChangePersonality(oldPersonality,
oldPersonality, oldPersonality);

    elseif(newCost(1) == oldCost(1) && newCost(2) > oldCost(2))
            newPersonality = ChangePersonality(oldPersonality, 2, 1);
    elseif(newCost(1) > oldCost(1) && newCost(2) > oldCost(2))
            newPersonality = ChangePersonality(oldPersonality, 2, 1);
    elseif(newCost(1) < oldCost(1) && newCost(2) > oldCost(2))
            newPersonality = ChangePersonality(oldPersonality, 2, 1);

    elseif(newCost(1) == oldCost(1) && newCost(2) < oldCost(2))
            newPersonality = ChangePersonality(oldPersonality,
oldPersonality, oldPersonality);
    elseif(newCost(1) > oldCost(1) && newCost(2) < oldCost(2))
            newPersonality = ChangePersonality(oldPersonality, 2, 1);
    elseif(newCost(1) < oldCost(1) && newCost(2) < oldCost(2))
            newPersonality = ChangePersonality(oldPersonality,
oldPersonality, oldPersonality);
    end
    paricle.Cost = ChangeCostValue(oldPersonality, newPersonality, oldCost,
newCost);
    paricle.Personality = newPersonality;
end
```

```
// ChangeCostValue.m

function value = ChangeCostValue(oldPersonality, newPersonality, oldValue,
newValue)
    if(oldPersonality == newPersonality)
        value = newValue;
    else
        value = oldValue;
    end
end
```

```
// ChangePersonality.m

function personality = ChangePersonality(current, trueValue, falseValue)
    if(current == 1)
        personality = trueValue;
    else
        personality = falseValue;
    end
```

```
// CreateGrid.m

function Grid=CreateGrid(pop,nGrid,alpha)

    c=[pop.Cost];

    cmin=min(c,[],2);
    cmax=max(c,[],2);

    dc=cmax-cmin;
    cmin=cmin-alpha*dc;
    cmax=cmax+alpha*dc;

    nObj=size(c,1);
    empty_grid.LB=[];
    empty_grid.UB=[];
    Grid=repmat(empty_grid,nObj,1);

    for j=1:nObj

        cj=linspace(cmin(j),cmax(j),nGrid+1);

        Grid(j).LB=[-inf cj];
        Grid(j).UB=[cj +inf];

    end

end
```

```
// DeleteOneRepMemebr.m

function rep=DeleteOneRepMemebr(rep,gamma)

    % Grid Index of All Repository Members
    GI=[rep.GridIndex];

    % Occupied Cells
    OC=unique(GI);

    % Number of Particles in Occupied Cells
    N=zeros(size(OC));
    for k=1:numel(OC)
        N(k)=numel(find(GI==OC(k)));
    end

    % Selection Probabilities
    P=exp(gamma*N);
    P=P/sum(P);

    % Selected Cell Index
    sci=RouletteWheelSelection(P);

    % Selected Cell
    sc=OC(sci);

    % Selected Cell Members
    SCM=find(GI==sc);

    % Selected Member Index
    smi=randi([1 numel(SCM)]);

    % Selected Member
    sm=SCM(smi);

    % Delete Selected Member
    rep(sm)=[];

end
```

```
// DetermineDomination.m

function pop=DetermineDomination(pop)

    nPop=numel(pop);

    for i=1:nPop
        pop(i).IsDominated=false;
    end

    for i=1:nPop-1
        for j=i+1:nPop

            if Dominates(pop(i),pop(j))
               pop(j).IsDominated=true;
            end

            if Dominates(pop(j),pop(i))
               pop(i).IsDominated=true;
            end

        end
    end

end
```

```
// Dominates.m

function pop=DetermineDomination(pop)

    nPop=numel(pop);

    for i=1:nPop
        pop(i).IsDominated=false;
    end

    for i=1:nPop-1
        for j=i+1:nPop

            if Dominates(pop(i),pop(j))
               pop(j).IsDominated=true;
            end

            if Dominates(pop(j),pop(i))
               pop(i).IsDominated=true;
            end

        end
    end

end
```

```
// FindGridIndex.m

function particle=FindGridIndex(particle,Grid)
```

```
    nObj=numel(particle.Cost);

    nGrid=numel(Grid(1).LB);

    particle.GridSubIndex=zeros(1,nObj);

    for j=1:nObj

        particle.GridSubIndex(j)=...
            find(particle.Cost(j)<Grid(j).UB,1,'first');

    end

    particle.GridIndex=particle.GridSubIndex(1);
    for j=2:nObj
        particle.GridIndex=particle.GridIndex-1;
        particle.GridIndex=nGrid*particle.GridIndex;
        particle.GridIndex=particle.GridIndex+particle.GridSubIndex(j);
    end

end
```

```
// Mutate.m

function xnew=Mutate(x,pm,VarMin,VarMax)

    nVar=numel(x);
    j=randi([1 nVar]);

    dx=pm*(VarMax-VarMin);

    lb=x(j)-dx;
    if lb<VarMin
        lb=VarMin;
    end

    ub=x(j)+dx;
    if ub>VarMax
        ub=VarMax;
    end

    xnew=x;
    xnew(j)=unifrnd(lb,ub);

end
```

```
// RouletteWheelSelection.m

function i=RouletteWheelSelection(P)

    r=rand;

    C=cumsum(P);
    i=find(r<=C,1,'first');

end
```

```
// SelectLeader.m

function leader=SelectLeader(rep,beta)

    % Grid Index of All Repository Members
    GI=[rep.GridIndex];

    % Occupied Cells
    OC=unique(GI);

    % Number of Particles in Occupied Cells
    N=zeros(size(OC));
    for k=1:numel(OC)
        N(k)=numel(find(GI==OC(k)));
    end

    % Selection Probabilities
    P=exp(-beta*N);
    P=P/sum(P);

    % Selected Cell Index
    sci=RouletteWheelSelection(P);

    % Selected Cell
    sc=OC(sci);

    % Selected Cell Members
    SCM=find(GI==sc);

    % Selected Member Index
    smi=randi([1 numel(SCM)]);

    % Selected Member
    sm=SCM(smi);

    % Leader
    leader=rep(sm);

end
```

```
// PlotCosts.m

function PlotCosts(pop,rep)

    pop_costs=[pop.Cost];
    plot(pop_costs(1,:),pop_costs(2,:),'ko');
    hold on;

    %rep_costs=[rep.Cost];
    %plot(rep_costs(1,:),rep_costs(2,:),'r*');
    xlabel('1^{st} Objective');
    ylabel('2^{nd} Objective');

    grid on;

    hold off;

end
```

```
functions used

// ZDT1.m

function z=ZDT1(p)

    x = p.Position;

    n=numel(x);

    f1=x(1);

    g=1+9/(n-1)*sum(x(2:end));

    h=1-sqrt(f1/g);

    f2=g*h;

    z=[f1
       f2];

end
```

```
// ZDT2.m

%% Zitzler-Deb-Thiele's test problem 2 (ZDT2) - (DEB, pag. 357)
 % data ---available
function z=ZDT2(p)

    x = p.Position;


    nx = 30;                        % 'n_x' states
    z1=x(1);
    z2=(1 + 9*sum(x(2:nx))/(nx-1))*(1 - (x(1)/(1 + 9*sum(x(2:nx))/(nx-
1))).^2);
    z=[z1 z2]';


end
```

```
// ZDT3.m
```

```
    %% Zitzler-Deb-Thiele's test problem 3 (ZDT3) - (DEB, pag. 357)
    %data ----available discrete P F
function z=ZDT3(p)

    x = p.Position;

    nx = 30;                            % 'n_x' states
    z1=x(1);
    z2=(1 + 9*sum(x(2:nx))/(nx-1))*(1 - sqrt(x(1)/(1 + 9*sum(x(2:nx))/(nx-
1))) - x(1)*sin(10*pi*x(1))/(1 + 9*sum(x(2:nx))/(nx-1)));
    z=[z1 z2]';




end
```

// Fonseca and Fleming function MOP2.m

```
function z=MOP2(p)

    x = p.Position;

    n=numel(x);

    z1=1-exp(-sum((x-1/sqrt(n)).^2));

    z2=1-exp(-sum((x+1/sqrt(n)).^2));

    z=[z1 z2]';

end
```