GENETIC ALGORITHM FOR PERSONNEL ASSIGNMENT PROBLEM WITH MULTIPLE OBJECTIVES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YILMAZ ARSLANOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JANUARY 2006

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

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

Prof. Dr. Ayşe Kiper
Head of Department

This is to 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.

Assoc. Prof. Dr. İsmail Hakkı Toroslu
Supervisor

Examining Committee Members

Prof. Dr. Faruk Polat	(METU, CENG)	
Assoc. Prof. Dr. İsmail Hakkı Toroslu	(METU, CENG)	
Prof. Dr. Nur Evin Özdemirel	(METU, IE)	
Assoc. Prof. Dr. Göktürk Üçoluk	(METU, CENG)	
Dr. Onur Tolga Şehitoğlu	(METU, CENG)	

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Yılmaz Arslanoğlu

Signature:

ABSTRACT

GENETIC ALGORITHM FOR PERSONNEL ASSIGNMENT PROBLEM WITH MULTIPLE OBJECTIVES

ARSLANOĞLU, Yılmaz

MS, Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. İsmail Hakkı Toroslu

January 2006, 68 pages

This thesis introduces a multi-objective variation of the personnel assignment problem, by including additional hierarchical and team constraints, which put restrictions on possible matchings of the bipartite graph. Besides maximization of summation of weights that are assigned to the edges of the graph, these additional constraints are also treated as objectives which are subject to minimization. In this work, different genetic algorithm approaches to multi-objective optimization are considered to solve the problem. Weighted Sum – a classical approach, VEGA - a non-elitist multi-objective evolutionary algorithm, and SPEA – a popular elitist multi-objective evolutionary algorithm, are considered as means of solution to the problem, and their performances are compared with respect to a number of multi-objective optimization criteria.

Keywords: Personnel Assignment Problem, Multi-Objective Optimization, Genetic Algorithms

ÖZ

ÇOKLU KRİTERLİ PERSONEL ATAMA PROBLEMİ İÇİN GENETİK ALGORİTMA

ARSLANOĞLU, Yılmaz
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. İsmail Hakkı Toroslu

Ocak 2006, 68 sayfa

Bu tez, personel atama problemine iki taraflı grafikte gerçekleştirilebilecek olası eşleştirmelere sınırlandırma getiren hiyerarşi ve takım kısıtlandırmaları ekleyerek, çoklu-kriterli bir türevini ileri sürmektedir. Grafiğin kenarlarına atanmış ağırlıkların toplamının ençoklanması kriterinin yanında, bu ek kısıtlamalar da enazlanması gereken kriterler olarak değerlendirilmektedir. Bu çalışmada, problemi çözmek için değişik çoklu-kriter genetik algoritma yaklaşımları gözönüne alınmaktadır. Klasik yaklaşım olan Ağırlıklandırılmış Toplam, seçkinci olmayan bir evrimsel algoritma olan VEGA ve popüler bir seçkinci evrimsel algoritma olan SPEA probleme çözüm yöntemleri olarak düşünülmüş, başarımları birtakım çoklu-kriter değerlendirme ölçütleri bakımından mukayese edilmiştir.

Anahtar Kelimeler: Personel Atama Problemi, Çoklu Kriter Eniyileme, Genetik Algoritma

To my parents and teachers…
(in the 22nd year of my studentship)

ACKNOWLEDGMENTS

For his guidance, support, encouragement and understanding, I would like to express my deepest gratitude to my thesis supervisor Assoc. Prof. Dr. İsmail Hakkı Toroslu, being a student of whom is definitely a great chance and opportunity.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Genetic Algorithm (GA) is a popular and successful means of solving optimization problems. In this work, a genetic algorithm solution is sought for *the personnel assignment problem with hierarchical ordering and team constraints,* which is a multi-objective extension of the personnel assignment problem.

Personnel assignment problem with hierarchical ordering and team constraints is a real life problem, which appears in personnel assignment of large hierarchical organizations, such as military. In such hierarchical organizations, the personnel to be assigned form a level graph reflecting the rank structures of personnel, and the positions to be filled form a forest, where each tree in the forest represents a set of hierarchically related positions. The matching should satisfy the hierarchical ordering constraints represented on two partitions, which means a person assigned to a superior position should have a higher rank. Also, the nodes of both partitions can form mutually exclusive sets, where all members of a set in the personnel partition are expected to be matched with members from the same set in the positions partition. On the personnel partition, these sets may represent families or teams that must be assigned together to the positions at the same location, and, on the positions partition, these sets may represent the locations of the positions. Finally, the weights on the edges connecting the personnel set to the positions set represent the expected performance of personnel on jobs.

Personnel assignment problem with hierarchical ordering and team constraints is a multi-objective optimization problem. Besides the objective of maximizing the sum of weights assigned to the edges of the bipartite graph, the hierarchical ordering and team constraints can also be treated as objectives which are subject to minimization. In order to solve this multi-objective optimization problem, three methods, each of which represent a different multi-objective evolutionary algorithm approach are employed. *Weighted Sum* is a classical approach, which combines the multiple objectives into a single one. *VEGA*, on the other hand, represents a non-elitist multi-objective evolutionary algorithm, which returns a set of trade-off solutions, instead of a single one. Finally,

SPEA is an elitist multi-objective evolutionary algorithm, which keeps track of and regularly updates a set of non-dominated solutions that have been encountered throughout the evolution process.

In the second chapter, the personnel assignment problem and its variations are explained. In chapter 3, working principles of genetic algorithms are explained in detail. Since the assignment problem is permutation based, special crossover techniques for permutation based chromosomes are also introduced in this chapter. Multi-objective optimization concepts are introduced in the fourth chapter. In the fifth chapter, the aspects of the personnel assignment problem as a GA problem are considered, and the three approaches, namely, Weighted Sum, VEGA and SPEA are explained in detail. Finally, in chapter 6, a detailed analysis of performances of these methods are presented, with respect to a number of multi-objective optimization criteria, and afterwards, the conclusion of this thesis work is asserted in chapter 7.

CHAPTER 2

PERSONNEL ASSIGNMENT PROBLEM

In personnel assignment problem, the possible ways of assigning a set of personnel to available positions from a set with the same cardinality are sought, while taking also into account the weights each personnel is given with respect to each position. The weight value of a <*personnel, position*> tuple may keep information about the eligibility or the desire of the personnel for the position in subject, the resulting profit that would be gained, or something else, according to the nature of the problem.

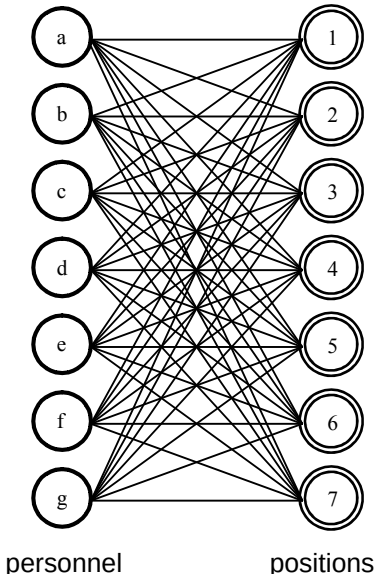


Figure-1 *Bipartite Graph – Personnel and Positions*

The problem can be illustrated as in Figure-1. As it can be seen from this figure, there are two mutually exclusive sets, one with the personnel to be assigned, and the other with positions, as their elements. Moreover, each personnel from the first set is connected to each position in the second set with an *edge*, which has a corresponding weight value. A sample weight assignment to edges can be seen in Table-1.

Table-1 *A Possible Weight Distribution on Edges*

	1	**2**	**3**	**4**	**5**	**6**	**7**
a	0	6	2	4	6	6	1
b	2	1	0	2	0	6	2
c	1	4	7	3	1	3	0
d	7	0	0	2	5	0	2
e	5	1	2	6	4	0	2
f	2	4	0	1	7	5	0
g	3	1	2	3	5	5	6

In this problem, the aim is to match each person from the personnel set with one of the elements from the positions set in such a way that, the sum of the weights of the corresponding edges resulting from this assignment will be the maximum available. Obviously, this problem is a bipartite graph matching, with weights on edges. In the literature, this is also known as the *standard assignment problem*, or *perfect maximum weighted bipartite graph matching*, which is a well known problem in graph theory. **[Mehlhorn & Näher 1999]**

2.1 Standard Assignment Problem

A *graph* G(V, E) is a data structure, which can be represented by a set of vertices V and a set of edges E, which connect any two vertices in V.

A *matching* M in a graph G = (V, E) is a subset of E such that, no two edges in E share a common vertex from V.

A matching M is called *perfect*, if it covers all vertices in V. That is, in order M to be perfect, for each vertex *v* in V, there should be an edge in M which covers *v* as one of its ending nodes.

A graph is called *bipartite* if and only if its vertex set V can be divided into two disjoint sets X and Y such that no two vertices from the same set are connected by an edge in E. Figure-1 constitutes an example to a bipartite graph.

A *maximum weighted matching* M in a graph G = (V, E) along with a weight function $w: E \rightarrow R$ is a matching where summing up $w(e)$ values for all $e \in M$ will result in the maximum possible sum, when applied over all possible subsets of E that represents a proper instance of matching.

From these definitions, *perfect maximum weighted bipartite graph matching* can be automatically defined. A perfect maximum weighted bipartite graph matching M in bipartite graph G = (V = (X, Y), E) is a subset of E such that:

- no two edges in M share a common vertex
- all of the vertices from X and Y are covered by an edge in M
- the edges in M start from and end with vertices from distinct sets
- the cardinalities of X and Y are equal.

According to this definition, the *standard assignment problem* automatically constitutes an instance of perfect maximum weighted bipartite graph matching. It can be represented according to this formalization:

- X is the set of personnel
- Y is the set of positions
- E is the set of edges that connect each personnel to each position
- Weight table (given in Table-1) is the weight function $w: E \rightarrow R$
- Finally, M is the <*personnel, position*> assignments, which is expected to maximize the *"profit"*.

In the literature of graph theory, there exists a number of ways to solve the perfect weighted bipartite graph matching problem in polynomial time. The most classical algorithm is due to Edmonds **[Edmonds 1965]**, which solves the problem in $O(V^3)$ time. This algorithm is based on the Hungarian method **[Kuhn 1955, Munkres 1957]**. More sophisticated and efficient algorithms can also be found in the literature. **[Galil 1986]**

2.2 Assignment Problem with Hierarchical Ordering Constraint

In the standard version of the assignment problem, there are no restrictions on possible matchings of the bipartite graph. That is, any vertex from X can be matched with a vertex in Y without any restriction. The only objective is to maximize the weights on edges. However, personnel assignment problem is a real-life problem encountered in hierarchical organizations such as military. In such a real-life problem, it is natural to expect constraints on possible matchings. For example, as a military organization, the Turkish Armed Forces assign thousands of personnel to vacant positions every year, and want to utilize the personnel to a maximum extent by assigning the right person to the right job, while taking into consideration the *hierarchy constraint*. **[Dinc & Oguztuzun 1998, Cimen 2001]**

According to this variation of the assignment problem, there is a hierarchy among the elements of both the set of personnel and the set of positions.

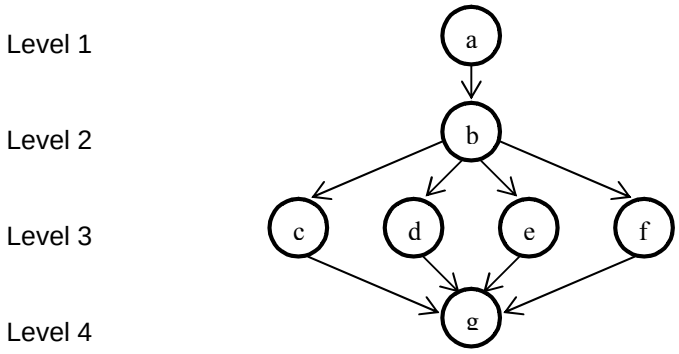


Figure-2 *Level Graph - Hierarchy Levels Among Personnel*

The hierarchy among the personnel of military is obvious: as it can be seen from the *level graph* in Figure-2, there are some levels in military organizations, where every person can be a member of exactly one of the levels (ranks) in the level graph, and a member of an upper level *dominates* all members in lower levels.

The set of positions also has a hierarchical structure among its elements (Figure-3). Intuitionally, this may be interpreted as a hierarchy in the case of a general manager, managers and their sub-workers.

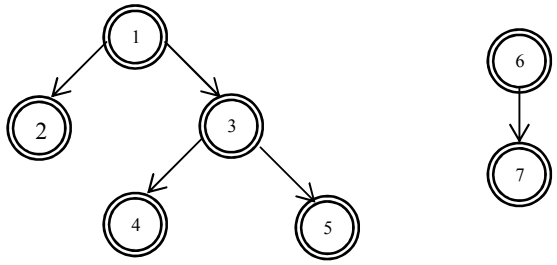


Figure-3 *Hierarchy Among Positions*

However, in the case of positional hierarchy, as opposite to personnel hierarchy, there does not exist the notion of levels. Instead, the hierarchical structure is represented as a *tree*, where a position *only* dominates the positions that could be reached by following its branches. Since, obviously, there may be irrelevant positions which do not originate from the same single root, we may have a number of trees (a *forest*) instead of a single tree structure.

- A person *A* dominates the person *B* (*A* > *B*) if and only if the rank of *A* is higher than the rank of *B*.
- A person *A* is equivalent to the person *B* (*A* = *B*) if and only if they share the same level.
- A position *Q* dominates the position *R* (*Q* > *R*), if and only if, following the branches, the node that represents the position R on the positional hierarchy tree could be reached, starting from the node that represents the position *Q* on the tree.

According to these definitions, the concept of *hierarchical violation* may be introduced as follows:

The number of hierarchical violations in matching M is the number of edge pairs $<e_1, e_2>$ in M, where the starting vertex (position) v_1 of e_1 dominates the starting vertex v_2 of e_2 ($v_1 > v_2$), but the ending vertex (person) y_1 of e_1 *does not dominate* or *is dominated by* the ending vertex y_2 of e_2 ($y_2 > y_1$ or $y_2 = y_1$). That is, a general cannot be assigned to a position that is supposed to serve a position occupied by a private.

Along with the bipartite graph and weights, having also the personnel ranks and positional hierarchy structure in hand, this version of the assignment problem is known as the *Assignment Problem with Hierarchical Ordering Constraint*, APHOC, due to Toroslu **[Toroslu 2003]**. In this natural variation of the standard assignment problem, the objective remains the same: finding a perfect matching on the bipartite graph that maximizes the weights. The actual distinction in this version is introduced by the single constraint of *minimizing the number of hierarchical violations*.

In his work, Toroslu proves the NP-completeness of this variation of the problem and proposes an efficient approximation algorithm.

2.3 Assignment Problem with Hierarchical Ordering and Team Constraints

In real life, when a person is assigned to a position, he / she must relocate to the city in which the position is located. Sometimes, there may be married couples in the personnel set. Therefore, in the assignment task, it also becomes necessary to assign such couples to positions that are located in the same city.

In this extension of the problem, while trying to maximize the weights, this new constraint along with the hierarchical ordering constraint also needs to be taken into account. This new version of the problem is called *Assignment Problem with Hierarchical Ordering and Team Constraints*. The new constraint is called *the team constraint*, because, the concept of couples is generalized to teams, in order to add more challenge to the problem.

According to this new version of the personnel assignment problem, there must be additional location information for positions. Moreover, there is a number of teams, each of which has a number of members greater than two. All members of a team must be assigned to positions such that they will be able to live in the same location.

In this thesis work, a genetic algorithm solution is sought for the Assignment Problem with Hierarchical Ordering and Team Constraints.

2.4 Constraints and Objectives

In an optimization problem, generally a set of constraints accompanies the problem, which restricts the whole search space of decision variables to a subset of it. If a solution that is found does not fall into this restricted region, such a solution is called *infeasible* and is not accepted as a solution. This kind of constraints that affect the acceptability of a solution are called *hard constraints*. However, there may also be a number of constraints that should be respected, but which will not affect the acceptability of the solution. Such constraints are called *soft constraints*.

In the assignment problem with hierarchical ordering and team constraints, these constraints are stated as follows:

- The solution should not lead to hierarchical violations
- The solution should not lead to team violations

There is also a single objective function:

- The solution should maximize the weight total

However, if these constraints are treated as hard constraints, in most of the problem instances, the search space will be tiny or almost an empty set, which will lead to infeasibility. For this reason, the problem is redefined as follows:

- The solution should maximize the weight total
- The solution should minimize hierarchical violations
- The solution should minimize team violations

According to this scheme, the constraints are also treated as objectives, which are subject to minimization.

CHAPTER 3

GENETIC ALGORITHMS

The main idea in the *Evolution Theory* of Charles R. Darwin is the *survival of the fittest*, which is also known as *natural selection* **[Darwin 1859]**. According to this theory, in a population of living things, fitter *generally* have a better chance to stay alive. Therefore, they, and their offspring who inherit their genetic content partially or completely from their parents, have a higher probability to go into next generations, and thus have a higher chance to transfer their genetic material to individuals which will appear in successor generations. *Genetic Algorithms*, which is also known as *Evolutionary Algorithms*, is a well known and widely accepted local search algorithm, which tries to simulate this theory. **[Holland 1975]**

3.1 Basic Concepts

```
GeneticAlgorithm (Integer populationSize,
                  Probability reproductionProbability,
                  Probability mutationProbability)
begin
      create Population p of populationSize
            and populate it with randomly generated individuals;

      while stopping condition not met do
      begin
            evaluate the fitness of individuals in population p;

            // start new generation

            create an empty Population temp of populationSize;

            // populate new generation

            while capacity of temp is not full do
            begin
                  assign Boolean doReproduction randomly,
                  according to reproductionProbability;

                  if doReproduction is TRUE then
```

```
                    begin
                          select an individual from p
                          and assign it to Individual firstParent;

                          select an individual from p
                          and assign it to Individual secondParent;

                          reproduce firstParent, secondParent
                          and get Individual newIndividual;
                    end
                    else
                          select an individual from p
                          and assign it to
                          Individual newIndividual;

                    assign Boolean doMutation randomly,
                    according to mutationProbability;

                    if doMutation is TRUE then
                          mutate newIndividual;

                    insert newIndividual to population temp;
              end;

              discard population p;
              set p as temp;
        end;
end;
```

Figure-4 *Basic Outline of Genetic Algorithms*

A genetic algorithm explores a number of potential solutions in parallel. It initially creates a *population*. A population is a set of *individuals*, each of which has its own genetic content: *chromosome*. In genetic algorithms, this genetic content, which is represented as a string over a finite alphabet, corresponds to a potential solution instance to the problem, and is coded according to a problem-specific coding scheme. In the initial population creation process, the genetic contents of individuals, that is, chromosomes, are generally produced in a randomized fashion in order to assure diversity in the initial population. Afterwards, in a loop of *evolution*, individuals and their offspring are transferred to new generations, taking into consideration the quality of their chromosomes, which is called *fitness*. Fitness is the objective function which takes an individual as an argument, evaluates its eligibility as a solution to that problem by examining its genetic content (chromosome), and assigns a fitness value to the individual as a result. The better fitness value gives to an individual a better chance to be selected for survival or reproduction.

An individual that exists in the current generation may be selected directly, or it may be matched with another individual and the resulting offspring may be transferred to the next generation.

Reproduction probability is the parameter that determines what percentage of the new generation will be composed of the individuals directly transferred from the previous generation, and what percentage will be composed of newly produced offspring.

A genetic algorithm should stop the simulation of the evolution process at some point. It may be terminated after exceeding a predetermined time threshold, producing a predetermined number of generations or reaching a desired fitness threshold. However, a genetic algorithm is generally terminated when it *converges*. Convergence occurs when most of the individuals in a population have very similar genetic content. In some situations, this may happen very rapidly so that it becomes impossible to reach to the optimal solution. This problem, which is called *premature convergence*, is similar to the problem of getting stuck on a local maximum that is encountered in local search methods. In order to overcome the problem of premature convergence, another natural mechanism which is observed in evolution can be employed: *mutation*.

During the transfer of genetic contents from parents to offspring, something may go wrong, and random changes may occur in chromosome. Such changes may also occur when an individual is exposed to extreme conditions such as radiation. As a result, the fitness value of the individual may degrade considerably. However, this helps to keep diversity in population and such changes may lead to very good results in the next generations.

In genetic algorithms, the concept of mutation is employed by introducing a *mutation probability* parameter. Before inserting an individual to the next generation, random changes on its chromosome is performed according to this probability.

The effect of mutation can be visualized in the following way:

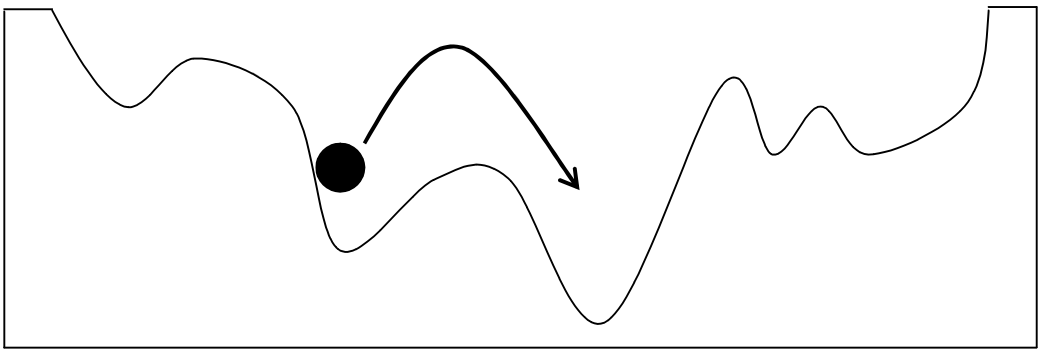


Figure-5 The Effect of Mutation

Assume that there is a ball and a box with the given shape in Figure-5. The aim is to make the ball go to the deepest location (global minimum) of this box. However, there is only a hole on the top of the box to let the ball in, and the actual shape of the surface inside cannot be seen. If the ball is let from the hole, it can go to a local minimum and stop there. If the box is shaken, the ball can dislocate from the local minimum and can have a chance to find its way to the global minimum. But if the box is shaken too hard, the ball can dislodge from the global minimum, if it was already located there.

In a similar way, in order to prevent premature convergence, mutation can be employed (shaking the box), but it must be done rarely enough, so that the population can finally converge to a solution.

3.2 Chromosome Representation in GA

In genetic algorithms, each individual that is a member of the population represents a potential solution to the problem. This solution information is coded in the associated chromosome of that individual. A chromosome is a string of *gene* positions, where each gene position holds an *allele* value that constitutes a part of the solution to the problem. Allele value at a gene position represents an element from a finite alphabet. This alphabet depends on the nature of the problem. There is a number of possible chromosome representations, due to a vast variety of problem types. However, there are two representation types which are most commonly used: *binary representation* and *permutation representation*.

3.2.1 Binary representation

The most common chromosome representation used in genetic algorithms is the *binary representation*. In this representation, the finite alphabet domain which each allele at each gene position takes its value from is the set {0, 1}. If the chromosome length is fixed (which is generally true), each chromosome will be a string of $\{0, 1\}^n$, where n represents the chromosome length.

1	0	1	1	0

Figure-6 *Binary Chromosome Representation*

The chromosome given in Figure-6 may hide a completely different information, depending on the problem. If the problem is concerned with finding the maximum value of the function *f(x):* {0, 31} $\rightarrow$ R, then the chromosome given in Figure-6 would be representing the decimal value of 22.

3.2.2 Permutation Representation

In the Travelling Salesman Problem (TSP), there are a number of cities, where each pair of cities has a corresponding distance **[Lawler et al. 1985]**. The aim is to visit all the cities such that the total distance travelled will be minimum. Obviously, a solution, and therefore a chromosome which represents that solution to the TSP, can be given as an order, that is, a permutation, of the cities.

4	1	3	5	2

Figure-7 *Permutation-Based Chromosome Representation*

The chromosome given in Figure-7 represents a solution such that the cities should be visited in this order: fourth, first, third, fifth and second.

3.3 Selection Mechanism

To mate the individuals for reproduction to create new offspring, or to transfer a part of the existing population to the next generation, we need a mechanism to select individuals. It is possible to perform the task of selection completely in a randomized fashion, which is called *uniform selection*. This selection mechanism will eventually cause the algorithm reach the global maximum. However, according to this scheme, convergence of the population will almost be impossible, and termination will take a considerably long time.

The main advantage of genetic algorithms comes from being able to calculate the eligibility of individuals as a potential solution to the problem, producing better generations at each time by making use of these fitness values, and thus converging to an optimal solution in the search space. For this reason, most selection mechanisms make use of the fitness values of individuals. Some of the most preferred selection schemes are the *elitist selection*, *roulette wheel selection* and *tournament selection*.

3.3.1 Elitist Selection

Elitism is a general concept and there exists a number of ways to employ elitism in genetic algorithms **[Deb 2001]**. One of the ways to realize this is to favor the top individuals and to ignore the remaining ones. According to this selection scheme, individuals in the population are sorted according to their fitness values. The best *n* individuals are included in the selection process and the remainings are discarded. The selection among the best *n* individuals is realized just as in the way it is done in uniform selection. That is, each individual that belongs to top *n* has a probability of (1 / n) of being selected. The pseudocode given in Figure-8 illustrates the mechanism:

```
ElitistSelection (Population p, Integer n) : returns an Individual
begin
      sort population p according to fitness values
      of individuals in descending order;

      assign Integer i a random number from the range [0, n-1];

      return the ith individual of population p;
end;
```

Figure-8 *Elitist Selection*

This selection method is widely used for its contributions in the speed of convergence, because of obvious reasons. However, it should be used carefully, in order not to encounter premature convergence.

3.3.2 Roulette Wheel Selection

Consider a roulette wheel with a number of slices on it, each of which has an associated width (Figure-9).

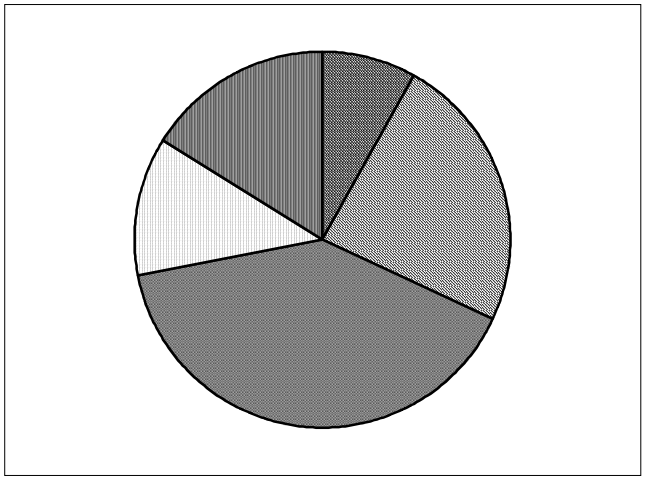


Figure-9 *Roulette Wheel*

If a ball is put on this wheel and the wheel is rotated, the ball will finally stop on one of the slices, most probably on one of the widest ones. However, all slices have a chance, with a probability that is proportional to its width. Roulette wheel selection attempts to simulate this behaviour. The pseudocode given in Figure-10 illustrates this mechanism:

```
RouletteWheel (Population p) : returns an Individual
Begin
      // first of all, calculate the sum of fitness values of all
      // individuals, if it is not known beforehand

      initialize Real fitnessSum as 0.0;

      for all Individual ind of population p, increase
            fitnessSum with the fitness value of ind;

      // randomly assign a value to determine
      // how much the wheel will turn

      assign Real threshold a random turn amount by
            fitnessSum * (a random real number from range [0.0, 1.0]);

      // find the partition where the ball on the wheel will stop

      initialize Real cumulative as 0.0;

      for each Individual ind of population p do
      begin
            increase cumulative by the fitness value of ind;

            if cumulative >= threshold then
                  return ind;
            else
                  continue with other individuals in p;
      end;
end;
```

Figure-10 *Roulette Wheel Selection*

Obviously, this selection mechanism cannot be used directly with a genetic algorithm where negative fitness values are allowed. In order to employ roulette wheel selection in such situation, a *transformation* over the fitness values can be applied.

The basic advantage of roulette wheel selection is in the way that it discards none of the individuals in the population and gives a chance to all of them to be selected. Therefore, diversity in the population is preserved. That is, the individuals other than the best ones also have the chance to transfer their genetic content to next generations, some of which may be hiding very valuable alleles. If it were not so, none of the gamblers would put their money into danger by selecting a slice on the wheel other than the widest one.

3.3.3 Tournament Selection

In tournament selection technique, *n* individuals are selected randomly from the population and the one with the highest fitness value is returned. The parameter *n* represents the tournament size. The pseudocode given in Figure-11 illustrates this selection technique:

```
TournamentSelection (Population p, Integer tournamentSize) :

                              returns an Individual
begin
      // select an individual randomly as the current winner

      assign Integer i a random number in the range
                        [0, (size of p) - 1];

      initially set Individual winner as the ith individual
                        of population p;

      // at each remaining tournament step, select an individual
      // randomly. If it is better, update the the winner

      initialize Integer tournamentStep as 1;

      while tournamentStep < tournamentSize do
      begin
            assign Integer i a random number in the range
                        [0, (size of p) - 1];

            if fitness value of the ith individual of population
            p is better than fitness of the current winner then
                  change winner as the ith individual of p;

            increment tournamentStep by 1;
      end;

      return winner;
end;
```

Figure-11 *Tournament Selection*

In the pseudocode given in Figure-11, an individual can be compared with itself, although this may sound weird in a real tournament. However, among all randomness in GA processes, this situation can simply be ignored.

Again, as in the case of roulette wheel selection, tournament selection also gives a chance to all individuals to be selected and thus it preserves diversity, although keeping diversity may degrade the convergence speed. However, diversity may be favored up to some extent because of its obvious advantages.

3.4 Reproduction Mechanism

As it can also be seen in the pseudocode of the main body of a genetic algorithm given in Figure-4, newly produced offspring constitute a part of the new generation, according to the given *ReproductionProbability* parameter. According to this probability, two parents are selected at each time and their chromosomes are *blended* and assigned to the new offspring. This blending process is performed according to a *Reproduction Mechanism*, which is called *crossover* in biology.

In the nature, although it may be much more complicated, crossover basically occurs as follows: Chromosomes of both parents are randomly divided from the same gene positions into a number of segments and the corresponding segments are exchanged and copied to the chromosome of the newly created offspring. Therefore, the offspring inherit traits from both parents.

In genetic algorithms, using binary string representation, the crossover process can be visualised as follows:

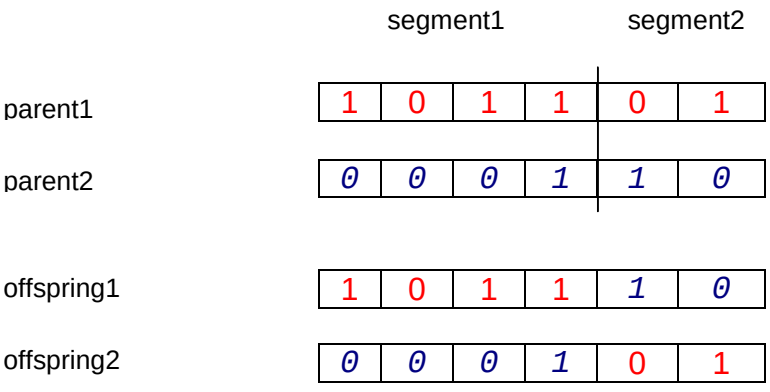


Figure-12 *One-Point Crossover*

This crossover technique is called *one-point crossover*, because the chromosomes are divided into two segments at a single gene position. Variations of this technique such as two-point (can also be generalized to *n*-point) also exist in the literature.

3.4.1 Permutation Based Crossover Techniques

If classical crossover techniques are applied on a permutation-based chromosome,

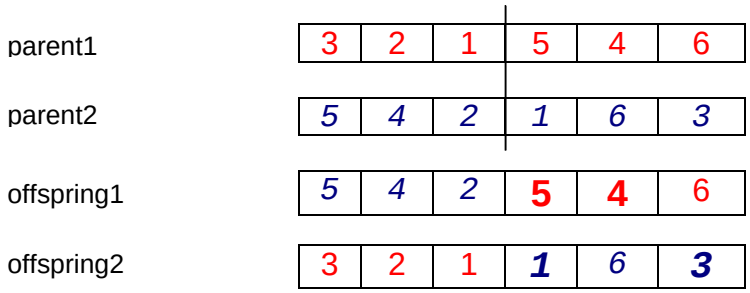
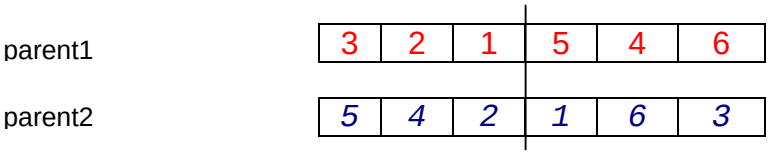


Figure-13 *One-Point Crossover on a Permutation-Based Chromosome*

invalid offspring may result, as illustrated in Figure-13. Such crossover techniques cannot preserve permutations. For this purpose, special techniques for permutation-based chromosomes are deployed, which ensure that, when applied on two permutation-based chromosomes, the chromosomes of the resulting offspring are also valid permutations. Beyond problem-specific special crossover methods, *Partially Mapped Crossover (PMX)*, *Ordered Crossover (OX)* and *Cycle Crossover(CX)* are the most popular generic permutation-based crossover techniques used in genetic algorithms. **[Goldberg & Linge 1985, Davis 1985, Oliver et al. 1987]**

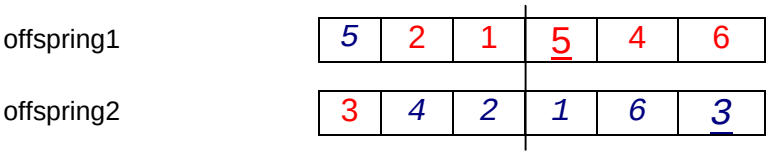
3.4.1.1 Partially Mapped Crossover (PMX)

The main purpose of a crossover operator is to create offspring that inherit traits from both parents. *Partially Mapped Crossover (PMX)* tries to obey this rule as much as it can, and it basically works like traditional crossover techniques like one-point crossover, while repairing damages instantly, that may occur on the permutation structure during the crossover process.

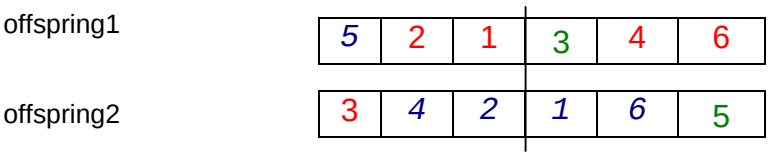


On these two parent chromosomes, the allele values at the first three gene positions will be swapped.

Starting from the first gene position, values are swapped:

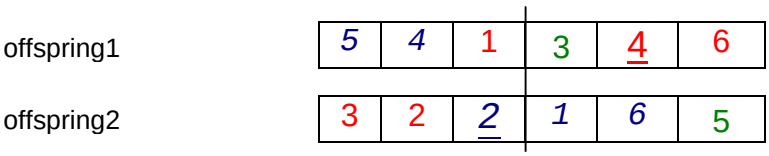


and the damage, emphasised with underlines, is repaired in a straightforward way:

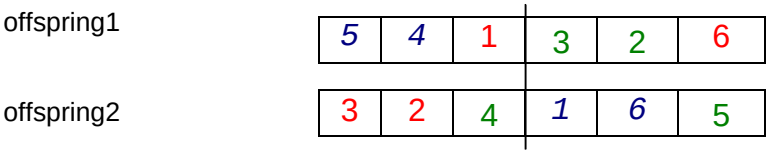


The allele values in the second and the third gene positions are swapped and the damages are repaired in a similar way:

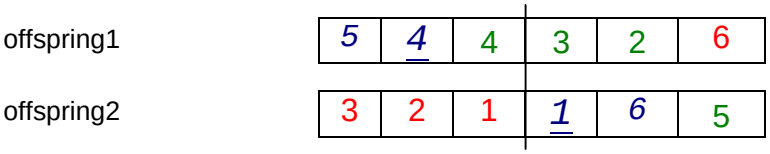
swap allele values at the second gene positions:



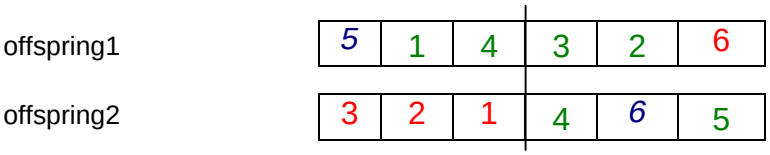
repair:



swap allele values at the third gene positions:



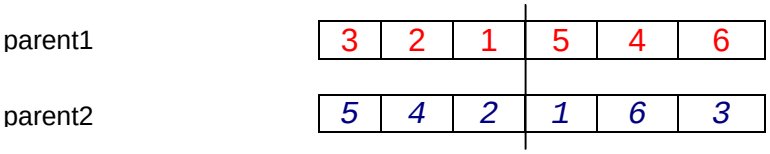
repair:



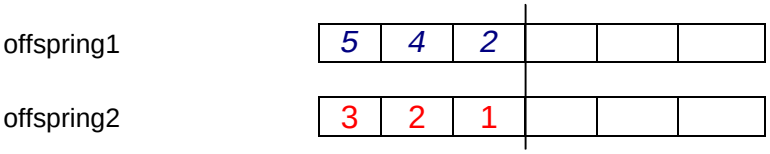
PMX tries to inherit traits from both parents as much as possible, but it cannot escape from introducing new traits that exist in the genetic content of none of the parents.

3.4.1.2 Order Crossover (OX)

In some permutation-based problems like TSP, relative order of allele values may be more important than absolute allele values at gene positions. Besides keeping traits from parents, *Order Crossover (OX)* takes also this aspect into account while performing the crossover operation. The following example illustrates how OX works:

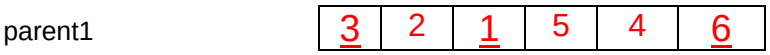


Firsly, selected segments are exchanged:



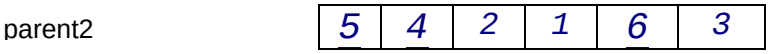
Then, empty gene positions of the first offspring are filled with the remaining possible values (1, 3, 6)

using the relative order in the first parent:

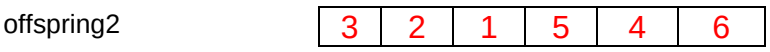


the first offspring is:

Similarly, empty gene positions of the second offspring are filled with the remaining possible values (4, 5, 6) using the relative order in the second parent:



the second offspring is:



As in the case of PMX, OX may also introduce traits that come from none of the parents (as it can be seen in offspring1, shown with a different font in boldface), but it preserves some of the relative orderings from either parent.

3.4.1.3 Cycle Crossover (CX)

In both PMX and OX, it is possible to introduce traits that exist in none of the parents. *Cycle Crossover (CX)* is designed to guarantee that each allele value at each gene position of the offspring chromosomes comes from one of the parents.

parent1	6	3	10	2	1	4	7	9	8	5
parent2	8	1	2	10	6	9	4	7	3	5

As opposite to PMX and OX, the chromosomes are not splitted to form segment(s) to swap. Just starting from the first gene position from the first parent, the following procedure is applied till the end:

For the first offspring, the allele value at the first gene position of the first parent is copied:

offspring1	6									

The second offspring has a single chance for its first gene position, and that is the allele value at the first gene position of parent2. Because, otherwise an additional trait will be introduced:

offspring2	8									

Since the allele value 8 for the second offspring is fixed, this value should be kept intact in the first offspring, according to the rule.

offspring1	6								8	

Similarly, the allele value 3 in offspring2 should be kept at the same gene position:

offspring2	8								3	

This continues in the same way and the first cycle (when the allele value that was first started with is reached) is completed:

offspring1	6	3			1				8	

offspring2	8	1			6				3	

The second cycle starts with with the first allele value that has not yet been considered in parent2 (at each cycle the parent to start with should change). The same applies in the rest of the cycle:

offspring1	6	3	2		1				8	
offspring2	8	1	10		6				3	
offspring1	6	3	2	10	1				8	
offspring2	8	1	10	2	6				3	

The third cycle starts with the first untouched allele value of parent1:

offspring1	6	3	2	10	1	4			8	
offspring2	8	1	10	2	6	9			3	
offspring1	6	3	2	10	1	4		9	8	
offspring2	8	1	10	2	6	9		7	3	
offspring1	6	3	2	10	1	4	7	9	8	
offspring2	8	1	10	2	6	9	4	7	3	

Finally, the last cycle starts with the second parent:

offspring1	6	3	2	10	1	4	7	9	8	5
offspring2	8	1	10	2	6	9	4	7	3	5

As it can be seen from the resulting offspring, each allele value at each position comes from one of the parents.

3.5 Mutation Mechanism

In order to abstain from getting stuck onto a local maximum, population diversity is required to be kept up to some extent. In genetic algorithms, this is achieved by the help of a mutation mechanism, which causes some sudden changes on the traits of individuals, although generally rarely, according to a predefined mutation probability parameter.

In binary chromosome representation, such changes can be done by inverting the allele value of a random gene position from 0 to 1 (and vice versa). In permutation-based representation, this can be achieved by swapping the allele values at two randomly chosen gene positions.

CHAPTER 4

MULTI-OBJECTIVE OPTIMIZATION PROBLEMS

In a *multi-objective (or multi-criteria) optimization problem*, there are more than one objective function to be taken into consideration. Besides the existence of multiple objective functions, there may also be some *constraints*, which put restrictions on the search space.

Here is a formalization of multi-objective optimization problem:

optimize	$f_i(\mathbf{x})$ for all i = 1, 2, ..., I
with	$g_j(\mathbf{x}) \leq 0$ for all j = 1, 2, ..., J
and	$h_k(\mathbf{x}) = 0$ for all k = 1, 2, ..., K.

In this formalization, **x** represents the solution vector (a point in the search space) that holds the *decision variables*, the function set *f* represents the objective functions subject to optimization, the function set *g* contains the inequality constraints and finally the function set *h* contains the equality constraints.

In the single-criterion case, generally there exists a global maximum, and the aim of a search algorithm is to reach to that peak point. However, in the existence of multiple objectives, this is generally not the case. As in almost all real life problems, the objective functions which are expected to be optimized are generally in conflict, that is, it is impossible to find out such a single point which will cover optimal values with respect to all objective functions.

The following example illustrates the situation (the example is from the book of Kalyanmoy Deb) **[Deb 2001]**:

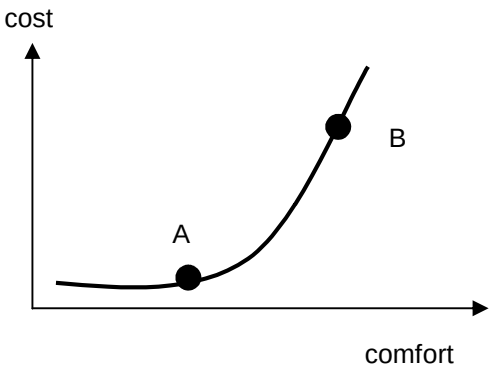


Figure-14 *Buying a Car – Cost vs Comfort*

Assume that there is a need to buy a car. As it is the case that everybody would like to select the one that is cheap and as comfortable as possible, one would like to maximize the comfort and minimize the cost of the car. However, if some trivial exceptions are ignored, it is generally not possible to find such a car that is both very cheap and comfortable. As it can be seen in Figure-14, since a preference must be done between A and B, one has to sacrifice one of the objectives, depending on subjective preferences. If A is chosen, the cost is minimized, but there is a loss in comfort, however in choice B, there is a gain in comfort but this time more money should be paid.

As the given example suggests, it would be possible to find out points that would represent the optimal value regarding each objective independently. However, it would generally not be possible to find an optimal solution vector which will favor all objectives. That is, in multi-objective optimization problems, there is actually no "optimal" solution. Instead, there is a set of possible *trade-off* solutions, among which the end user is expected to make a selection according to personal preferences. Therefore, there is a need for redefinition of the concept of optimality for the case of multiple objectives: *Pareto Optimality*. Before that, the concept of *dominance* should be introduced:

4.1 Dominance

A solution vector **x** is said to *dominate* the solution vector **y** when:

- $\forall i \in \{1, 2, ..., I\}$ such that $f_i(\mathbf{x}) \geq f_i(\mathbf{y})$ and
- $\exists\, i \in \{1, 2, ..., I\}$ such that $f_i(\mathbf{x}) > f_i(\mathbf{y})$.

That is, **x** dominates **y** when it is as good as **y** regarding each objective, and there is at least one objective with respect to which **x** is better than **y**.

4.2 Pareto Optimality

A solution vector $\mathbf{x} \in X$ is *Pareto-Optimal*, if there is no solution vector in the search space X that dominates **x**.

Such pareto-optimal solutions form the *Pareto-optimal Set*. A sample pareto-optimal set of a problem with two objective functions subject to minimization is illustrated in Figure-15:

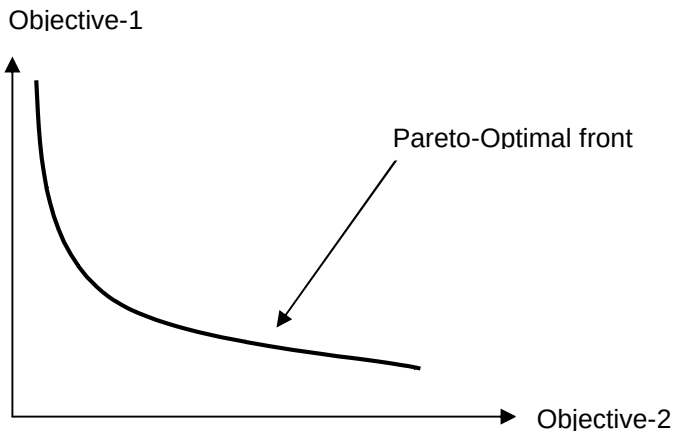


Figure-15 *Pareto-Optimal Front*

The curve shown in Figure-15 is called the *Pareto-Optimal Front*, which holds the elements of the pareto-optimal set. The points that fall onto the inside region of this curve form the *attainable set*, elements of which represent sub-optimal solutions, which are dominated by some solutions on the search space. However, the points on the pareto-optimal front can be dominated by none of the solutions in the search space, and thus it is not possible to enhance one of the objectives on this curve without sacrificing the other.

Figure-16 illustrates possible pareto-optimal fronts for problems with two objectives, with respect to their optimization configurations regarding minimization or maximization:

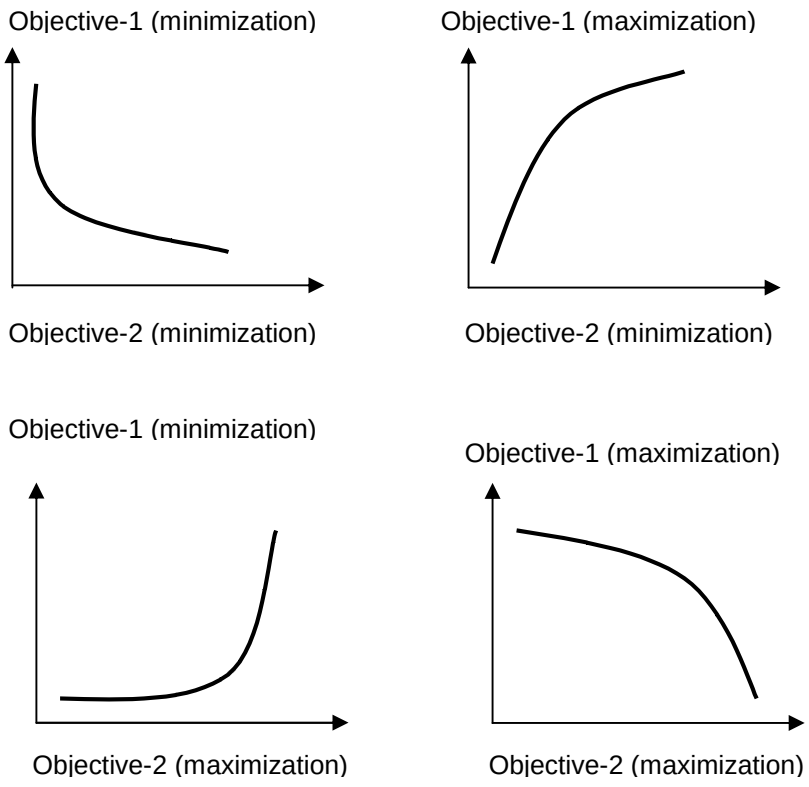


Figure-16 *Example Pareto-Optimal Fronts for a 2-Objective Problem*

Clearly, the aim of a solution mechanism for a multi-objective optimization problem should be finding as large of a trade-off solution set as possible, whose elements are scattered close to or just on the pareto-optimal front. That is, in order to be able to present to the end user a diverse set of possible non-dominated trade-off solutions, they need to be chosen on the pareto-optimal front, or at least close to it. Figure-17 illustrates an ideal set of trade-off solutions in this respect:

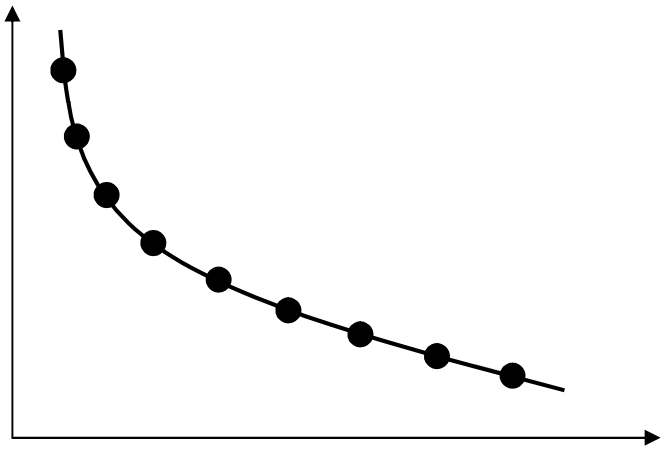


Figure-17 *Ideal Distribution of Trade-Off Solutions*

CHAPTER 5

PERSONNEL ASSIGNMENT AS A GA PROBLEM

5.1 Chromosome Representation

The core of the *assignment problem with hierarchical ordering and team constraints* is matching a set of personnel to a set of positions, where the cardinality of these two sets are equal. Since each personnel can be assigned to exactly one position, a potential solution can be represented as an ordering of personnel (an ordering of positions is also possible):

8	1	10	2	6	9	4	7	3	5

Figure-18 *Chromosome Representation of Personnel Assignment Problem*

This chromosome holds the following information: person with id 8 is assigned to the first position, person with id 1 is assigned to the second position, and so on.

From this perspective, it seems best to use a pure permutation representation because of the nature of the problem and the simplicity of the representation, although some other permutation representation methods that allow traditional crossover and mutation techniques exist in the literature. **[Ucoluk 2002]**

5.2 Weighted Sum – A Classical Approach

The pioneering approach to solve a multi-objective optimization problem is the intuitive idea of collecting all objectives into a single one. There is a set of classical methods, which aggregate the

objective functions and try to find a single solution that represents the optimal of this single function. The most popular method that falls into this category is the *Weighted Sum*, which still presents a powerful means of solving multi-objective optimization problems. **[Liu et al. 1998, Jakob et al. 1992, Gen et al. 1995, Rubenstein-Montano & Malaga 2002]**

In this weighted approach, a single objective function from all objective functions is built, by giving each of them a weight, according to its importance. Each objective function is multiplied with its corresponding weight, and they are summed up to obtain the fitness value that is to be assigned to the individual.

Fitness =

*– teamWeight * (teamViolations / maximumPossibleTeamViolations)*
*– hierarchyWeight *(hierarchyViolations / maximumPossibleHierarchyViolations)*
*+ weightTotalWeight * (weightTotal / maximumPossibleWeightTotal)*

Figure-19 *Weighted Fitness*

- Some objectives may be subject to minimization and some others to maximization. In order to get around this complicacy, the *duality principle* is used, which states that maximization of an objective is equivalent to minimization of its negated form **[Rao 1984]**. In Figure-19, the overall objective function is preferred to be maximized. Therefore, the objective functions that are subject to minimization are negated.

- The magnitude of values of individual objective functions may differ considerably. For example, in a problem instance, the weight total may be expressed in thousands, whereas the number of team violations may be expressed in tens and the number of hierarchical violations in hundreds. Therefore, in order to abstain from biasing the solution towards an individual objective, these values should be *normalized*. In Figure-19, this is achieved by dividing the actual objective value to previously calculated maximum values, in order to get values on a scale of [0,1].

- The weight values that are assigned to individual objective functions should sum up to 1. That is, if 50 % importance is given to the minimization of hierarchical violations, and 30

% of importance is assigned to the maximization of weight total, the associated weight value of team violation minimization automatically becomes 20 %.

5.2.1 Advantages

- From a practical point of view, a user expects a single solution, instead of a set of solutions which has to be examined further. With this respect, this method is intuitive and easy to use.
- Since this method combines all objective functions into a single one, its implementation is straightforward using the basic genetic algorithm template.
- Since weight values can be assigned to individual objectives according to their importance, this method can serve as a genetic algorithm solution not only to the personnel assignment problem with hierarchical ordering and team constraints, but also to a collection of personnel assignment problems. That is, having this method in hand, the previous versions of the problem, namely, the standard assignment problem and the assignment problem with hierarcical ordering constraints, and also similar variations can be solved, using appropriate weight values.

5.2.2 Disadvantages

- Appropriate weight values are required to be known and set beforehand.
- Weighted Sum returns a single solution, instead of a set of trade-off solutions. In order to find out some other trade-off solutions, the algorithm needs to be run using different weight configurations.
- This method does not guarantee to reach all possible solutions on the pareto-optimal front, if the function that determines the pareto-optimal front of the problem is *non-convex*. **[Deb 2001]**

A function is convex, if and only if any two points on its curve can be connected with a line that falls completely inside the graph (Figure-20):

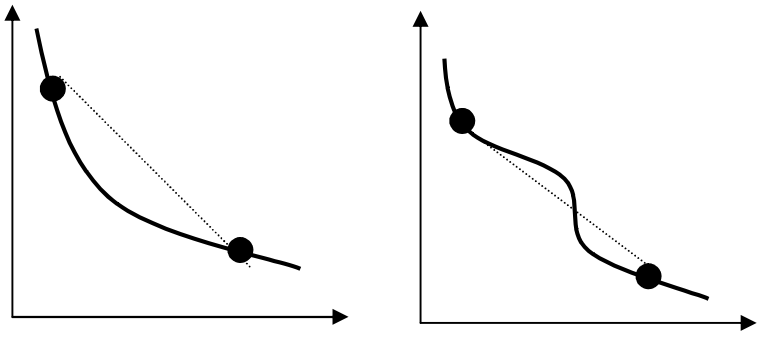


Figure-20 *Convex and Non-Convex Functions*

Figure-21 sketches the pareto-optimal solutions of a small sample problem with 8 personnels and positions (chromosome length is equal to 8) on a three dimensional objective space. As it can be seen from this figure, the pareto-optimal solutions are scattered in a randomized fashion and therefore does not represent a smooth surface, which inherently leads to the conclusion that the pareto-optimal front is non-convex. This means that, Weighted Sum does not ensure that all pareto-optimal solutions in the pareto-optimal set are reachable. That is, even if all possible weight configurations are tried, some pareto-optimal solutions may not be reached.

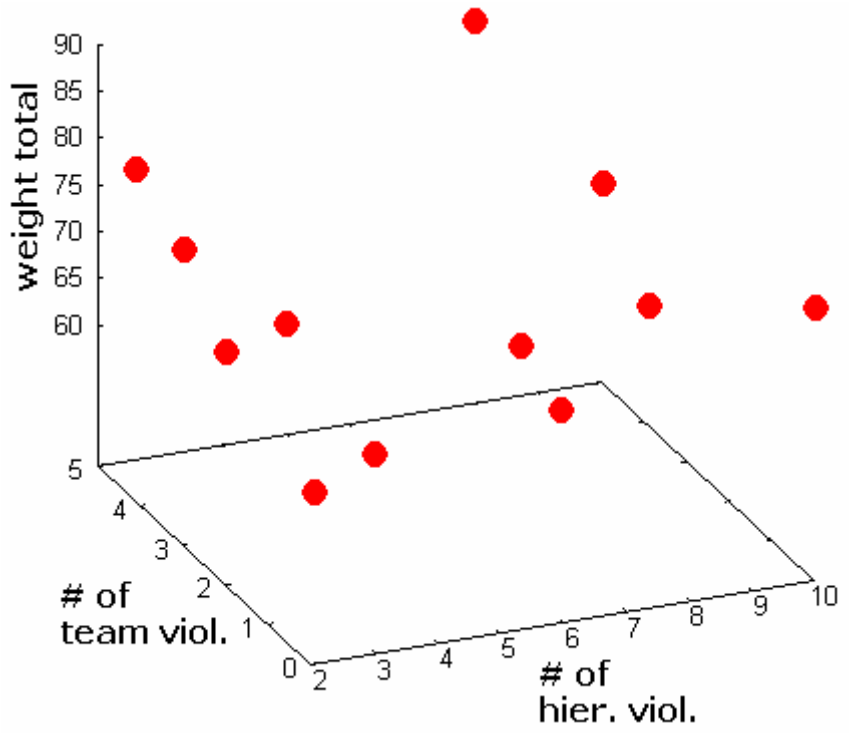


Figure-21 *Pareto-Optimal Front of a Small Problem*

5.3 Evolutionary Algoritms for Multi-Objective Optimization Problems

The main drawback of Weighted Sum is that, in each run, it can only find a single solution, instead of a set of trade-off solutions. There is also a need for setting an appropriate weight configuration beforehand. Furthermore, even run a number of times with different weight configurations, obtaining solutions from the entire pareto-optimal front is not guaranteed. In order to walk around such disadvantages, a number of algorithms are devised, which are called *Multi-Objective Evolutionary Algorithms*. This class of algorithms primarily takes advantage of the population-based mechanism of evolutionary algorithms. Instead of finding a single solution based on a single combined fitness function, they return a set of trade-off solutions. Furthermore, the user does not have to emphasize some objectives beforehand by using the weight values. Instead, the task of making a preference depending upon subjective criteria is deferred to the end of simulation.

5.3.1 Vector Evaluated Genetic Algorithm (VEGA)

The first algorithm that is proposed in order to abolish the drawbacks introduced by classical approaches is the *Vector Evaluated Genetic Algoritm*, *VEGA*, which was introduced by Schaffer **[Schaffer 1985]**. It is the simplest among a number of multi-objective evolutionary algorithms **[Coello 1999, Fonseca & Fleming 1995]**, because it can easily be adapted on the basic genetic algorithm template given in Figure-4 by only minor changes. At each generation, it divides the population into a number of equally sized sub-populations, each of which is associated with one of the objectives. Each individual in each sub-population is assigned a fitness value based on the corresponding objective function of that sub-population. Afterwards, a mating pool that has a size of the original population is created and is filled by applying the selection mechanism on each sub-population, where selection operation is restricted to individuals of that sub-population. Finally, crossover and mutation mechanisms are employed as usual, and the new population is generated. Figure-22 illustrates this scheme, applied on personnel assignment problem:

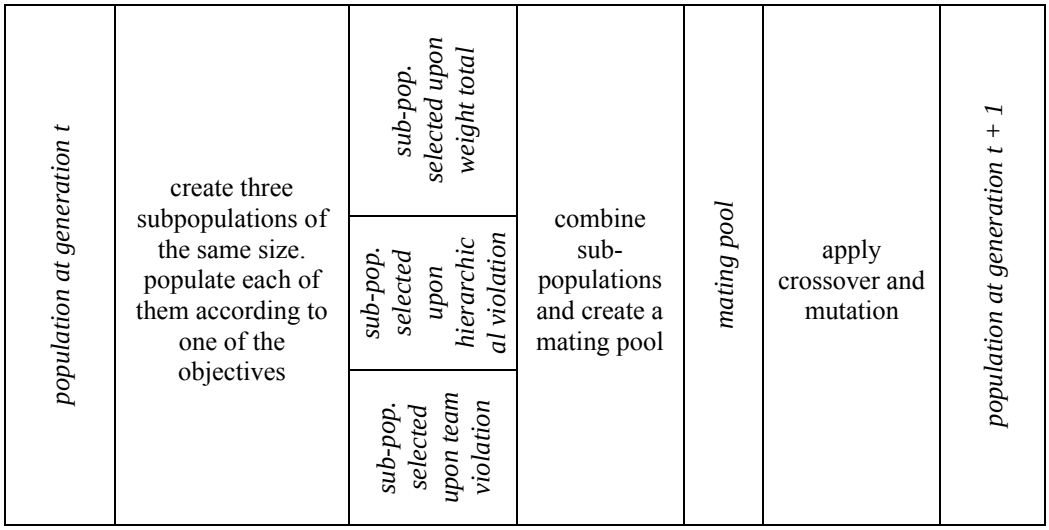


Figure-22 *Basic Outline of VEGA*

5.3.1.1 The Pseudocode

```
// generate the mating pool by selecting individuals depending
// on each objective function in turn

create an empty Population matingPool
      of capacity populationSize;

initialize Objective currentObjective as HIERARCHY_VIOL;

initialize Integer i as 0;

while i < populationSize do
begin
      select Individual ind from population p depending
      on currentObjective;

      insert individual ind to matingPool;

      if currentObjective is HIERARCHY_VIOL then
            set currentObjective as TEAM_VIOL;
      else if currentObjective is TEAM_VIOL then
            set currentObjective as WEIGHT_SUM;
      else
            set currentObjective as HIERARCHY_VIOL;

      increment i by 1;
end;
```

```
// populate new generation

create an empty Population temp of populationSize

while capacity of temp is not full do
begin
      assign Boolean doReproduction randomly,
      according to reproductionProbability;

      if doReproduction is TRUE then
      begin
            select an individual from matingPool
            and assign it to Individual firstParent;

            select an individual from matingPool
            and assign it to Individual secondParent;


            reproduce firstParent, secondParent
            and get Individual newIndividual;
      end
      else
            select an individual from matingPool
            and assign it to
            Individual newIndividual;

      assign Boolean doMutation randomly,
      according to mutationProbability;

      if doMutation is TRUE then
            mutate newIndividual;

      insert newIndividual to population temp;
end;

discard population p;
discard matingPool;
set p as temp;
```

Figure-23 *Pseudocode of VEGA*

As it can be seen from Figure-23, VEGA can easily be adapted on the basic genetic algorithm template by only doing a number of minor changes. Although it is the pioneering work in the field of multi-objective evolutionary algorithms, its main idea is used in some applications. **[Ritzel et al. 1994, Surry et al. 1995, Fourman 1985, Kursawe 1992]**

5.3.2 Strength Pareto Evolutionary Algorithm (SPEA)

One of the most reputable multi-objective evolutionary algorithm methods, which is also proven in **[Zitzler et al. 2000]** to perform better among a number of well known techniques is the *Strength*

Pareto Evolutionary Algorithm (SPEA) **[Zitzler & Thiele 1998]**. It falls into the category *Elitist Multi-Objective Evolutionary Algorithms*, because it introduces elitism by keeping track of and favoring a set of non-dominated (elite) solutions, which have been encountered throughout the evolution process. It achieves this task by introducing an additional external population P', which can contain up to a predefined number of non-dominated individuals, whereas the actual population P contains the ordinary individuals, as in the case of traditional GA approach. The elements of the external population P' are checked and updated regularly throughout the algorithm by inserting newly encountered non-dominated individuals and removing previously non-dominated individuals which are now dominated by some of newly generated ones. When the number of residents of P' exceeds a predefined value, it is shrunk according to a *clustering algorithm*, which at the same time tries to respect the second purpose of multi-objective optimization: preserving diversity. The members of P' are assigned *strength* values depending on the number of individuals from P that they dominate, and the members of P are assigned fitness values according to the number of individuals from P' that dominate them. After the assignment of fitness values, P and P' are considered as a single population, and usual GA operators are applied on this combined population. The SPEA is given in a step by step format in the following lines:

STEP 1: Generate an initial population *P* and create an empty external non-dominated set *P'*

STEP 2: Copy non-dominated members of *P* into *P'*, and at the same time, remove dominated members of *P'*.

STEP 3: If the size of *P'* exceeds a predefined size, prune it by using the clustering technique. The clustering technique partitions *m* elements group into *n* groups of relatively homogenous elements where $n<m$. The clustering technique in SPEA is as follows:

Initialization Phase: Construct cluster set *C* where each cluster contains one element of *P'*.

Iteration Phase: If the number of clusters is greater than *n*, calculate distances of all possible pairs of clusters where the distance of two clusters is defined as the average distance (such as Euclidian distance) between all pairs of individuals of the two clusters. Then, combine the two clusters with minimal distance, and, repeat this step.

Final Phase: If the number of clusters is equal to *n*, construct the new non-dominated set *P'* by selecting a representative individual from each cluster. The representative of a cluster is a member of the cluster, called as the *centroid*, with minimal average distance to all other members in the cluster.

STEP 4: Calculate the fitness of the individuals in *P* and *P'* as follows:

Phase 1 (P'): The fitness (also called as strength) of each member *i* of *P'* is defined as the ratio of the number of members in *P* dominated by *i* over the size of *P* plus 1 (the denominator is 1 plus the size of P, because, in SPEA fitness values are subject to minimization, and fitness values of elite

individuals should be less than 1, since fitness values of ordinary individuals will be guaranteed to be greater than one).

Phase 2 (P): The fitness of each member *i* of *P* is defined as the summations of the strengths of the members of *P'* that dominate *i* plus 1 (1 is added to the total, because the fitness values of ordinary individuals are desired to be greater than the strength of non-dominated individuals).

STEP 5: Select individuals from *P*+*P'* (multi-set union) in order to apply genetic operators by using the binary tournament. Then, apply crossover and mutation operators on this set as usual. If the maximum number of generations is not reached continue from STEP 2.

Notice that, unlike classical GA techniques, in SPEA smaller fitness values represent better solutions. Also, in SPEA the ratio of ¼ between *P'* and *P* is shown to be successful **[Zitzler et al. 2000]**.

5.3.2.1 The Pseudocode

```
// find non-dominated individuals of current population

set Population temp as the non-dominated
      individuals of population p;

// append it to the actual non-dominated set

append temp to nonDominatedSet;

// now, some of the individuals in nonDominatedSet may have
// been dominated. Remove them if there exists such solutions.

set nonDominatedSet as the
non-dominated set of itself;

if size of  nonDominatedSet >
            predefined nonDominatedSetSize then
      cluseter nonDominatedSet to nonDominatedSetSize;

// assign non-dominated fitness values

for each Individual ind in nonDominatedSet do
begin
      initialize Integer count as 0;

      for each Individual ind2 of population p do
            if ind dominates ind2 then
                  increment count by 1;

      set fitness of ind as (count / (populationSize + 1));
end;

// assign normal population fitness values

for each Individual ind in population p do
begin
      initialize Real sum as 1.0;

      for each Individual ind2 of nonDominatedSet do
            if ind2 dominates ind then
                  increment sum by fitness of ind2;

      set fitness of ind as sum;
end;

//  finally,  combine  P  and  P'  for  performing  usual  GA
operations

append nonDominatedSet to p;
```

Figure-24 Pseudocode of SPEA

CHAPTER 6

ANALYSIS

6.1 Performance Comparison of Weighted Sum, VEGA and SPEA

Weighted Sum, VEGA and SPEA were tried on a problem with the following characteristics:

Table-2 *Problem Parameters*

Parameter	Value
CHROMOSOME LENGTH	100
MAX. WEIGHT TOTAL	9900
MAX. HIERARCHICAL VIOL.	408
MAX. TEAM VIOL.	214

As the crossover technique, Cycle Crossover (CX) was used, since it can transfer the genetic content of parents to the maximum extent. As the selection technique, binary tournament (tournament selection with tournament size 2) was preferred. In the original work of VEGA [**Schaffer 1985**], roulette wheel selection is emphasized. However, in the tests performed, VEGA with roulette wheel led to a worse result. Therefore, using tournament with VEGA instead of roulette wheel when making such a performance analysis would not lead to an unfair result.

The GA parameters used for solution are as follows:

Table-3 *GA Solution Parameters*

Parameter	Value
CROSSOVER RATE	0.8
MUTATION RATE	0.1
POPULATION SIZE	100
EXTERNAL POPULATION SIZE (FOR SPEA)	25
NUMBER OF GENERATIONS	200
SELECTION METHOD	TOURNAMENT WITH SIZE 2
CROSSOVER METHOD	CYCLE CROSSOVER (CX)

These parameters are determined referring to mainly **[Zitzler et al. 2000]** and some other similar works.

Since Weighted Sum returns a single solution in a single run, it was tried on the same problem with 16 different weight configurations to obtain different trade-off solutions. Furthermore, the resulting non-dominated sets of VEGA and SPEA were clustered to size 16, using the clustering technique described in section 5.3.2, in order to have an equal set of solutions for all these three methods. Since clustering removes solutions that are close to existing ones, this will not have a direct affect on the results of the performance comparison.

6.1.1 Numerical Results

Table-4, Table-5, and Table-6 demonstrate the numerical results of Weighted Sum, SPEA and VEGA, respectively.

Table-4 *Weighted-Sum Numerical Results*

USED WEIGHT CONFIGURATION			*SOLUTION*		
HIERARCHICAL WEIGHT %	TEAM WEIGHT %	WEIGHT TOTAL WEIGHT %	HIERARCHICAL VIOLATIONS	TEAM VIOLATIONS	WEIGHT TOTAL
33.3	*33.3*	*33.3*	80	29	6507
50	*25*	*25*	66	32	6698
25	*50*	*25*	68	15	6107
25	*25*	*50*	87	70	7855
10	*45*	*45*	131	21	7267
45	*10*	*45*	52	132	7677
45	*45*	*10*	59	44	5688
40	*30*	*30*	67	53	7012
30	*40*	*30*	100	15	6526
30	*30*	*40*	79	55	7031
30	*35*	*35*	101	22	6840
35	*30*	*35*	88	36	6624
35	*35*	*30*	76	10	6765
100	*0*	*0*	39	175	5075
0	*100*	*0*	324	9	4693
0	*0*	*100*	254	175	8589

Table-5 *SPEA Numerical Results*

HIERARCHICAL VIOLATIONS	TEAM VIOLATIONS	WEIGHT TOTAL
68	94	6932
136	72	6930
77	136	7477
103	104	7449
165	145	7632
232	68	7054
169	99	7460
166	87	7289
82	89	7219
252	94	7379
95	121	7512
68	112	7133
86	99	7342
113	124	7629
77	127	7395
188	123	7705

Table-6 *VEGA Numerical Results*

HIERARCHICAL VIOLATIONS	TEAM VIOLATIONS	WEIGHT TOTAL
79	110	6424
78	114	6774
77	119	6762
76	112	6661
72	117	6498
73	112	6508
78	113	6757
98	117	6865
77	119	6762
77	119	6762
80	109	6649
77	119	6762
77	119	6762
77	119	6762
70	120	6675
77	112	6734

6.1.2 Multi-Objective Success Criteria

6.1.2.1 Average Euclidian Distance to the Utopian Objective Vector

Utopian Objective Vector is a non-existent solution, which is strictly better than any of the solutions in the search space. That is, for each objective, its corresponding value is better than that of the best solution that could be found in the search space with respect to this objective.

In personnel assignment problem, the utopian objective vector can be represented as follows:

Table-7 *Utopian Objective Values*

HIERARCHICAL VIOL.	TEAM VIOL.	WEIGHT TOTAL
0	0	MAXIMUM WEIGHT TOTAL

One of the two main purposes of multi-objective optimization algorithms is to find non-dominated solutions as close to the pareto-optimal front as possible.

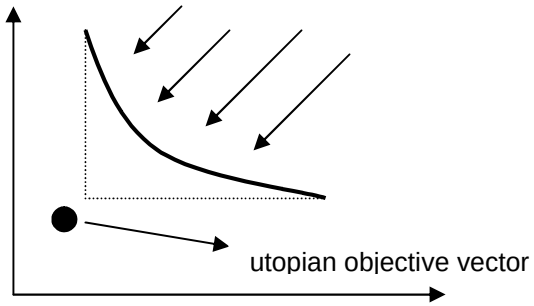


Figure-25 *Utopian Objective Vector*

Intuitionally, the average distance of the resulting non-dominated solutions to the utopian objective vector will reveal their closeness to the pareto-optimal front, as Figure-25 illustrates. However, this may not hold on non-convex pareto-optimal fronts when comparing individual solutions:

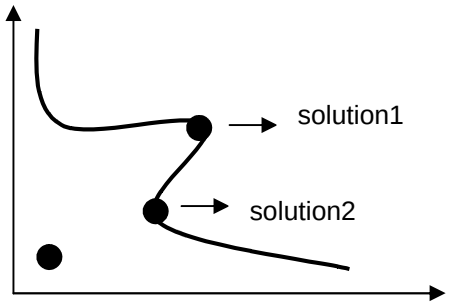


Figure-26 *Distance Comparison on Non-Convex Pareto-Optimal Front*

As it can be seen from Figure-26, although solution1 is further away from the utopian objective vector when compared to solution2, they are both on the pareto-optimal front and thus have the same acceptability as a solution.

However, since sets of solutions are compared instead of individual ones, the average distance of their elements can be compared. If solutions are well distributed, a fair comparison can be expected.

Making use of this idea, the individual distances of solutions found by Weighted Sum, VEGA and SPEA are calculated, and these distance values are plotted on a radar chart (Figure-27):

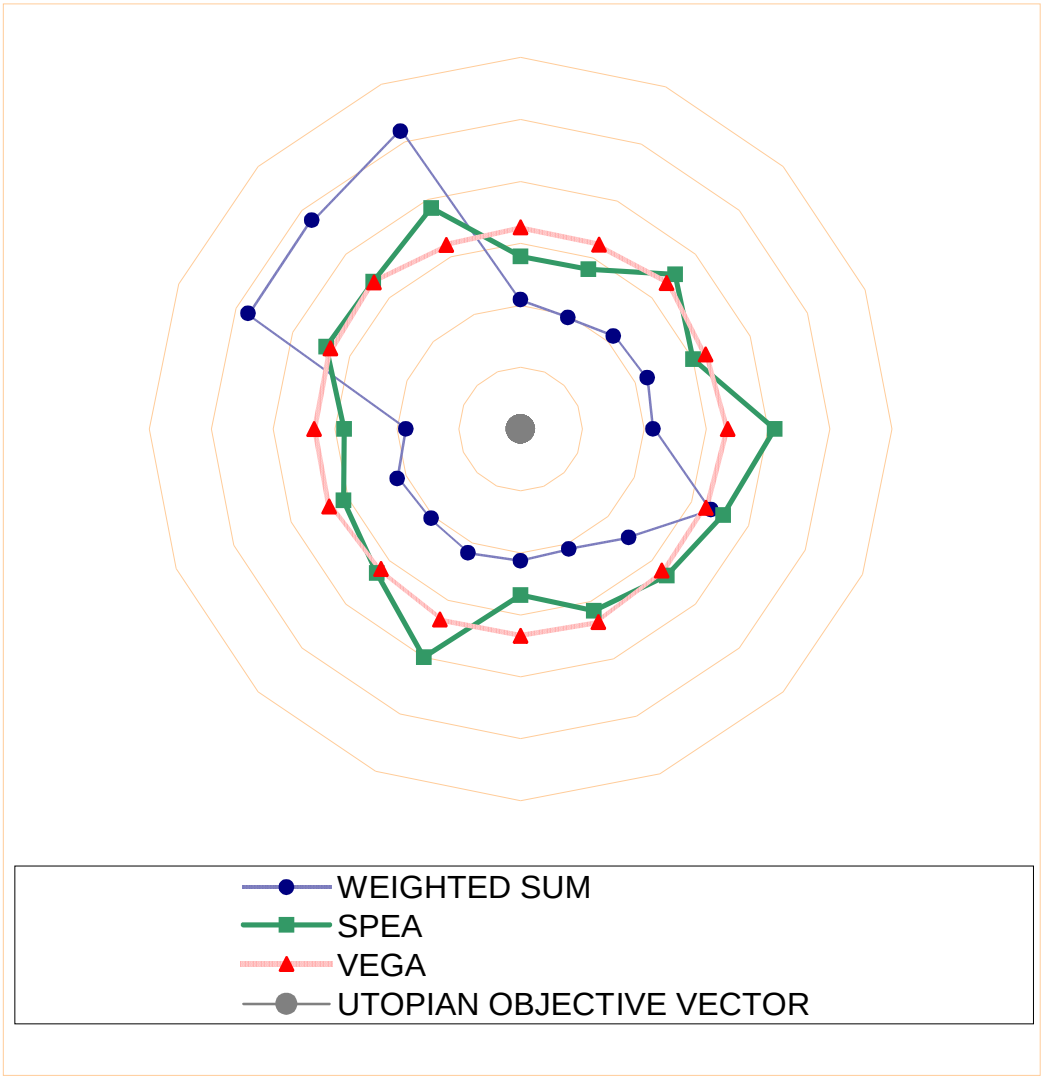


Figure-27 *Average Euclidian Distance to the Utopian Objective Vector*

In the radar chart given in Figure-27, the dot in the center represents the utopian objective vector. Each solution found by the three methods is plotted on this chart according to its euclidean distance from this vector. There are 48 such solutions on this chart. A set of 16 solutions represents the ones that are found by Weighted Sum, each of which is obtained by using a different weight configuration (these weight configurations are listed in Table-4). Another group of 16 solutions is the clustered set of 25 non-dominated solutions of SPEA (recall that non-dominated set size was chosen as 25). Finally, the remaining set of 16 solutions is obtained by clustering the non-dominated solutions of the final population of VEGA.

From this figure, it can be concluded that Weighted Sum returns the closest results to the pareto-optimal front. SPEA and VEGA result in similar distance averages. However, VEGA forms a uniform circle, which means that the set of solutions it returned are similar, not diverse. This can also be confirmed from Table-6. Again from Figure-27, it can be seen that SPEA returns a more diverse set of results than VEGA. However, by using Weighted Sum, such diversity can easily be obtained by arranging the weight values accordingly. Extreme values of Weighted Sum on this figure are such examples (the three solutions in the upper left corner), obtained by giving extreme weight configurations (last three rows in Table-4), which care only about a single objective and ignore the others.

6.1.2.2 Set Coverage Metric

Another metric which gives a clue about the closeness to the pareto-optimal front is the *set coverage metric*, which is proposed by Zitzler **[Zitzler 1999]**. Given two solution sets A and B, it simply counts the number of solutions in set B which are dominated by at least one of the solutions in set A, and returns the ratio by dividing this value by the cardinality of B.

$$\text{coverage}(A, B) = \frac{\text{cardinality}(b \in B \mid \exists\, a \in A : a \text{ dominates } b)}{\text{cardinality}(B)}$$

Figure-28 *Coverage Metric*

According to the coverage metric calculation given in Figure-28 and using the results given in Table-4, Table-5, and Table-6, the following coverage values are calculated:

```
coverage(WEIGHTED_SUM, SPEA) = 12 / 16
coverage(WEIGHTED_SUM, VEGA) = 16 / 16

   coverage(SPEA, WEIGHTED_SUM) = 0
    coverage(SPEA, VEGA) = 16 / 16

   coverage(VEGA, WEIGHTED_SUM) = 0
       coverage(VEGA, SPEA) = 0
```

Figure-29 *Calculated Coverage Values*

According to these coverage values, Weighted Sum performs the best, then comes SPEA, and finally comes VEGA.

6.1.2.3 Illustrative Representation

When there are two objectives, non-dominated solutions can easily be illustrated on a two-dimensional coordinate system. However, when three or more objectives are involved, a need for different illustration techniques emerges. There are a number of techniques proposed to illustrate non-dominated solutions of a problem with more than 2 objectives. They can be found in **[Meisel 1973, Cleveland 1994]**. However, the most popular, easiest and the most illustrative one is the *Value Path Method* **[Geoffrion et al. 1972]**.

In value path method, there are n vertical axes, each of which represent the associated scale of one of the objectives. On these vertical axes, each non-dominated solution's corresponding objective value is marked and these marked points of the solution are combined with straight lines, constituting a path for that solution.

The following value path graph illustrates the solutions obtained (from Table-4, Table-5, and Table-6):

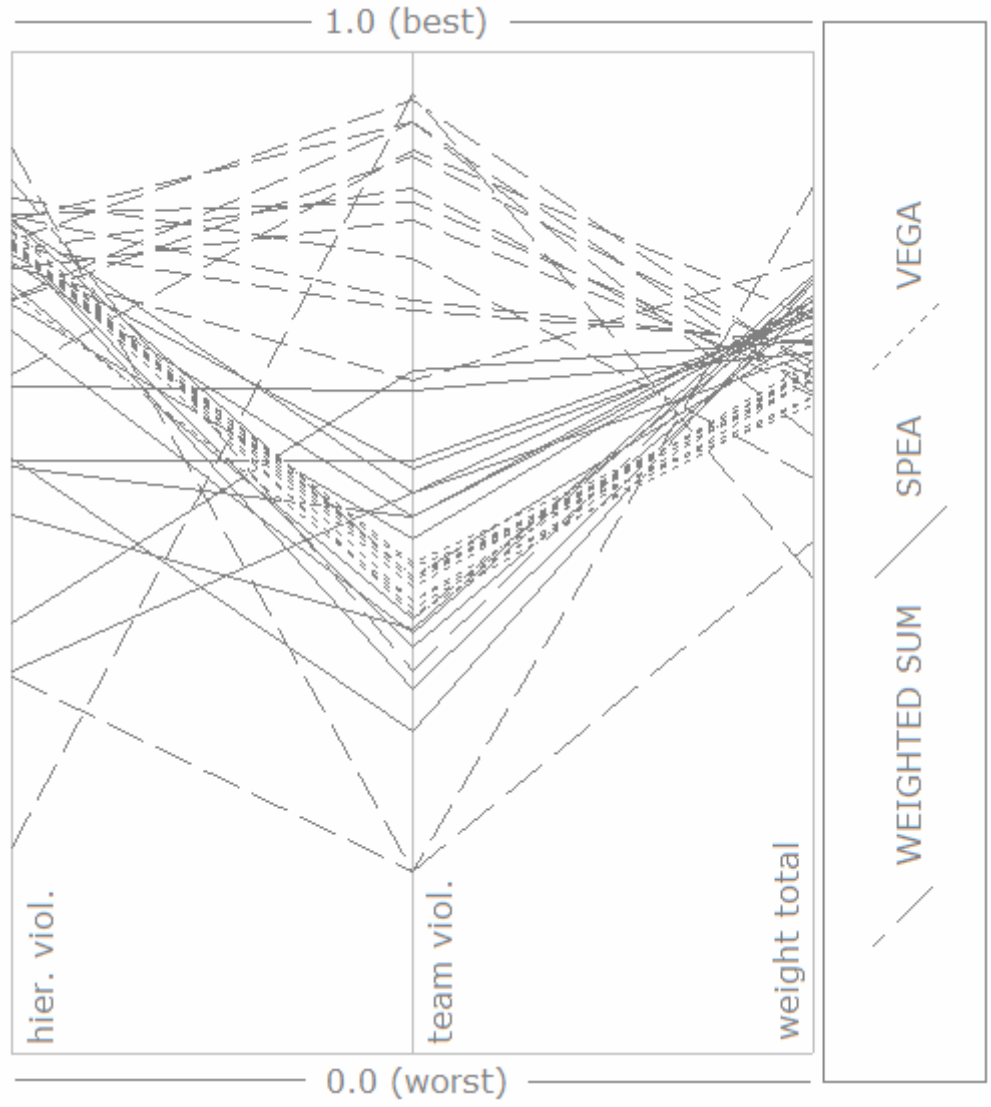


Figure-30 *Value Path Representation of Results*

When applying this illustration technique, the objective values were normalized by simply dividing them to their corresponding maximum values. Moreover, the objective values were rearranged in such a way that in Figure-30, a path which completely lies above another represents a better solution, regardless of whether the individual objectives are subject to minimization or maximization.

From the value path graph given in Figure-30, it can be concluded that Weighted Sum solutions dominate SPEA and VEGA solutions. Moreover, SPEA solutions dominate VEGA solutions. It is also possible to see that VEGA converged to a narrow region and does not present a diverse set of

solutions. SPEA is better than VEGA in diversity. However, obviously, Weighted Sum finds both a dominating and a diverse set of solutions.

6.1.3 Comments on Performance Comparison

Figure-21 illustrates the pareto-optimal front of a problem with chromosome length 8. This means that, the search space consists of 8! = 40320 discrete states. However, the pareto-optimal front consists of only 12 states, which is incredibly small compared to the whole search space. This value will get larger as the problem size increases, however, it would still be a small number of points concentrated close to the utopian objective vector, but non-uniformly distributed, because of the nature of the problem. In such a situation, it seems to be more logical to concentrate on this region and find at least one point that is close to one of these pareto-optimal solutions. Weighted Sum simply tries to do this. When such a single point can be found, it would most probably dominate the solutions that are found by a multi-objective evolutionary algorithm, which can also be inferred from Figure-29 and Figure-30.

Furthermore, the pareto-optimal front is non-uniformly distributed and highly non-convex. Because of non-uniformly distributed discrete points, it does not represent a smooth surface and it can be considered highly disconnected. All these characteristics of the personnel assignment problem constitutes major drawbacks which multi-objective evolutionary algorithms suffer from **[Deb 2001]**.

6.2 Parameter Configuration of Weighted Sum

Since the analysis results reveal that Weighted Sum performs better than the other two, it would be better to concentrate on Weighted Sum to improve its performance. This can be best realized by configuring the problem solution parameters. These parameters may include the followings:

- population size
- maximum number of generations (for stopping condition)
- crossover rate
- mutation rate

Along with these parameters, there is also a need to employ a selection and a reproduction (crossover) mechanism. All these parameter values and preferred selection and reproduction mechanisms form a *solution configuration*.

In order to find out the best solution configuration, Weighted Sum was used to solve three different problems, each with a chromosome length of 100 (cardinality of the position and the personnel set is 100), using 1650 different solution configurations, each of which is a combination of the parameter values and crossover and selection techniques, given in Table-8:

Table-8 *Problem Solution Configurations*

Parameter	CROSSOVER RATE	MUTATION RATE	CROSSOVER METHOD	SELECTION METHOD
Values	0.2	0.000	PMX	ELITISM 20 %
		0.002		
	0.4	0.004		ELITISM 40 %
		0.006		ELITISM 50 %
	0.5	0.008	CX	ELITISM 60 %
		0.010		ELITISM 80 %
	0.6	0.020		ROULETTE WHEEL
		0.040	OX	TOURNAMENT 2 %
	0.8	0.060		TOURNAMENT 4 %
		0.080		TOURNAMENT 6 %
		0.100		TOURNAMENT 8 %

- in Table-8, ELITISM X % means that selection is done among the top X % individuals.
- in Table-8, TOURNAMENT X % means that tournament size is the X % of the population size.

A population of size 100 was used and the algorithms were run 200 generations, regardless of checking whether convergence occured or not. After obtaining the results, the best 50 solution configurations among the 1650 were considered, and the occurences of each parameter value and crossover and selection technique in this top 50 solution configuration set were counted. The same thing was done for each of the three problems solved, and average of these occurrences were calculated.

6.2.1 Crossover Method

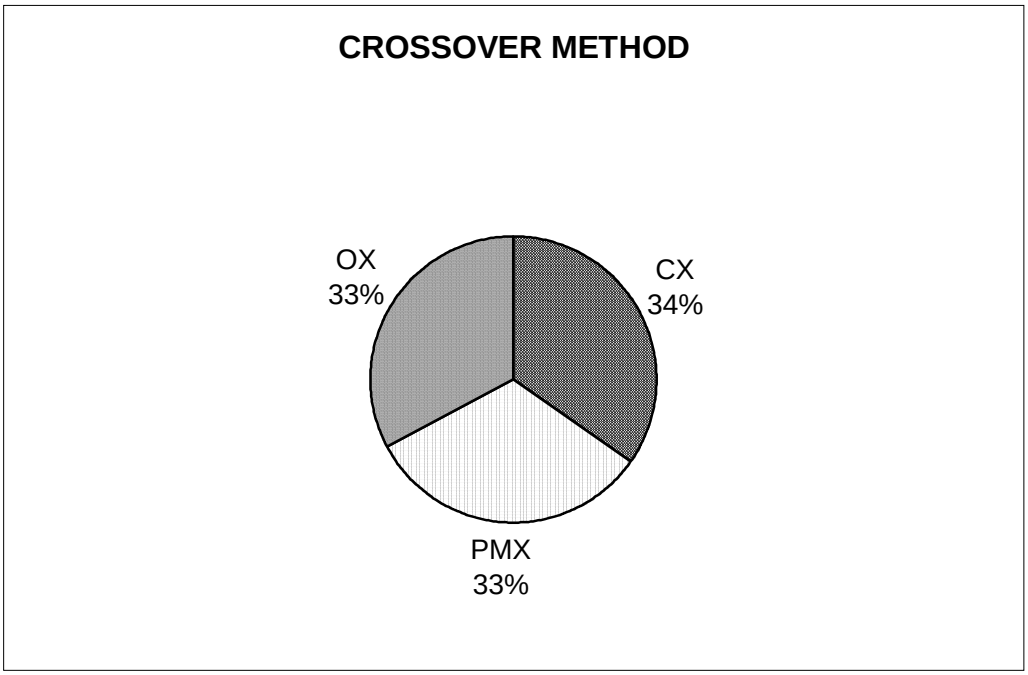


Figure-31 *Success of Crossover Methods on Weighted Sum*

As it can be seen from Figure-31, each crossover technique has almost the same effect. This situation results from the sensitivity of the problem to changes on permutation based chromosomes. Even a single swap of allele values may lead to a very high number of hierarchical violations or result in a perfect solution. Moreover, positive changes in one objective may affect other objectives negatively. Because of the nature of the problem and the chromosome structure, none of these permutation-based crossover methods can outperform the other. Some attempts to propose problem-specific crossover method did not turn out to be successful.

6.2.2 Selection Method

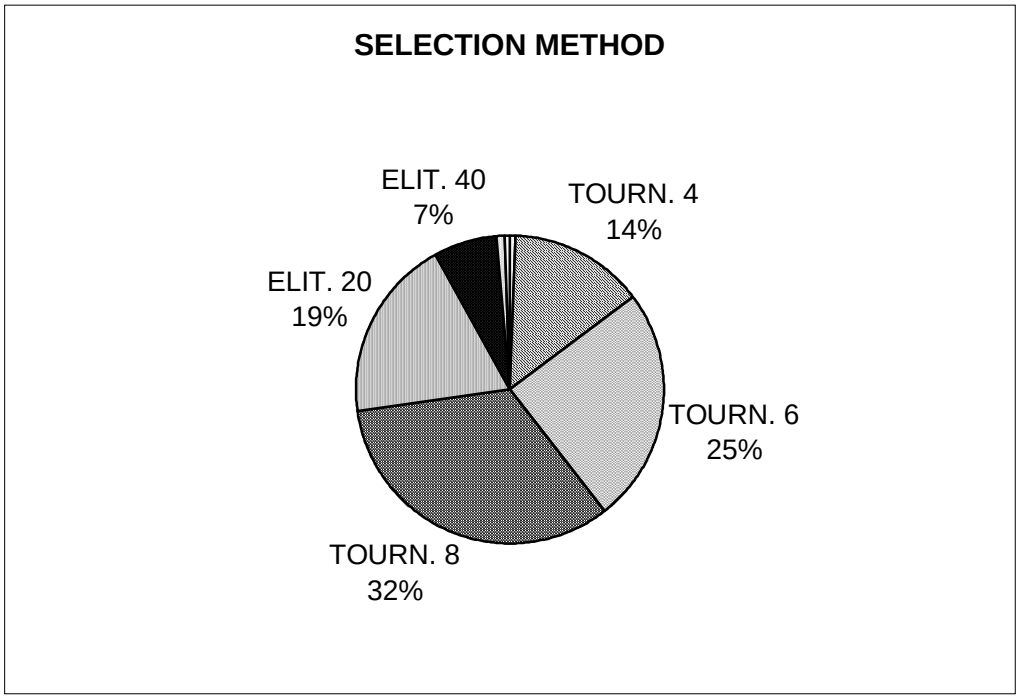


Figure-32 *Success of Selection Methods on Weighted Sum*

From the results Figure-32 illustrates, it can be concluded that, the concept of elitism has a drastical effect on the results. According to this figure, tournament selection with tournament size 8 performs the best and outperforms the elitism selection with a top 20 % selection range. However, it actually corresponds to an elitism selection with a top 12.5 % selection range. It is followed by tournament selection with size 6, which corresponds to an elitism selection with rougly top 16.5 % selection range. Then comes elitism selection with top 20 % selection range. It is followed by a tournament with size 4 (elitism 25 %) and finally comes elitism with top 40 % selection range. These results obviously show that elitism is favored to a great extent. Upon these results, further tests were done with elitism, and even much better results were obtained, when much more smaller top selection ranges were used.

6.2.3 Crossover Rate

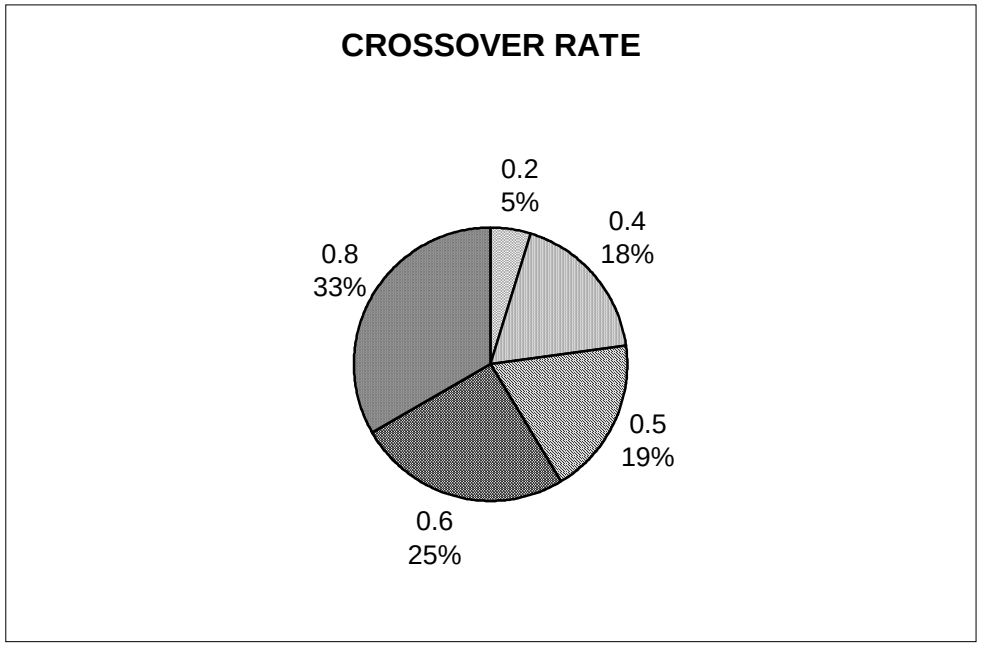


Figure-33 *Success of Different Crossover Rates on Weighted Sum*

From Figure-33, it can be concluded that the higher the crossover rate is, the better the results obtained. This means that, it would be better if newer offspring are produced instead of transferring the existing individuals to the next generation. In further tests where larger crossover rates were used, even better solutions were obtained.

6.2.4 Mutation Rate

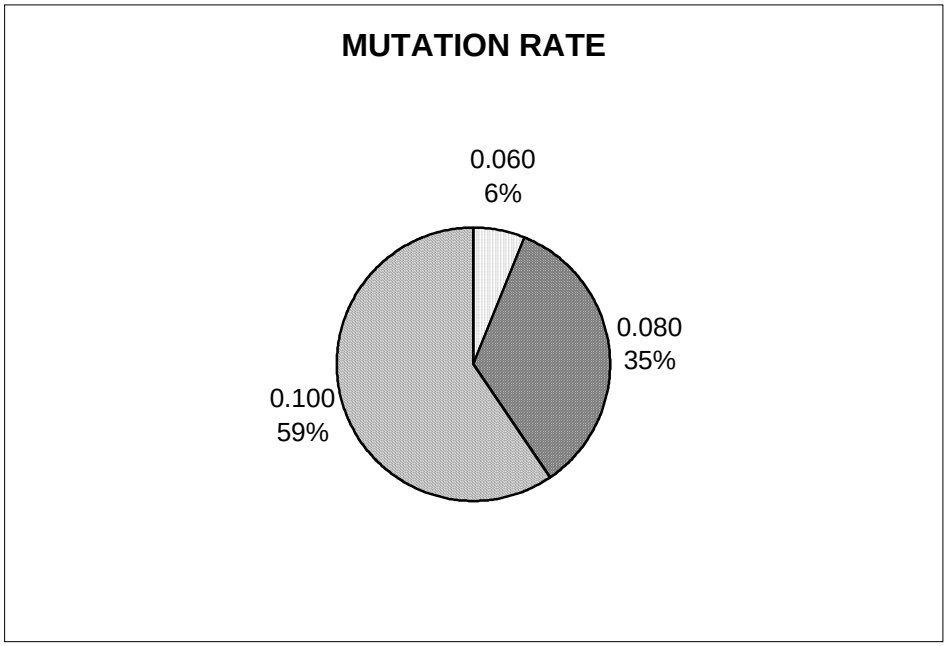


Figure-34 *Success of Different Mutation Rates on Weighted Sum*

Figure-34 shows that higher mutation rates lead to better results. In further tests, even higher mutation rates such as 100 % were applied and much better results were obtained.

Actually, Figure-32, Figure-33 and Figure-34 lead to a very consistent conclusion:

Selection is *exploitation*, while crossover and mutation is *exploration* **[Eiben & Schippers 1998]**. When there is too much exploitation, it may lead to premature convergence. However, in the existence of excessive exploration, this time, the algorithm cannot concentrate on a solution. Therefore, a genetic algorithm should find a trade-off between these two concepts. By elitism selection, the favorable treats of the good individuals are exploited. However, because of the specific structure of the landscape of the problem (Figure-35), an excessive exploration needs to be done, instead of allowing early convergence. Only by producing newer offspring and applying high rates of mutation, it would be possible to check a wide range of states on this landscape.

Figure-35 illustrates a part of the landscape of a small problem. This sample problem has 8 positions and 8 personnel to be assigned. The graphic is obtained by enumerating all possible permutations (8! = 40320) by a recursive permutation generation algorithm, calculating their respective weighted fitness values and taking the part that covers the first 200 permutations of the whole search space:

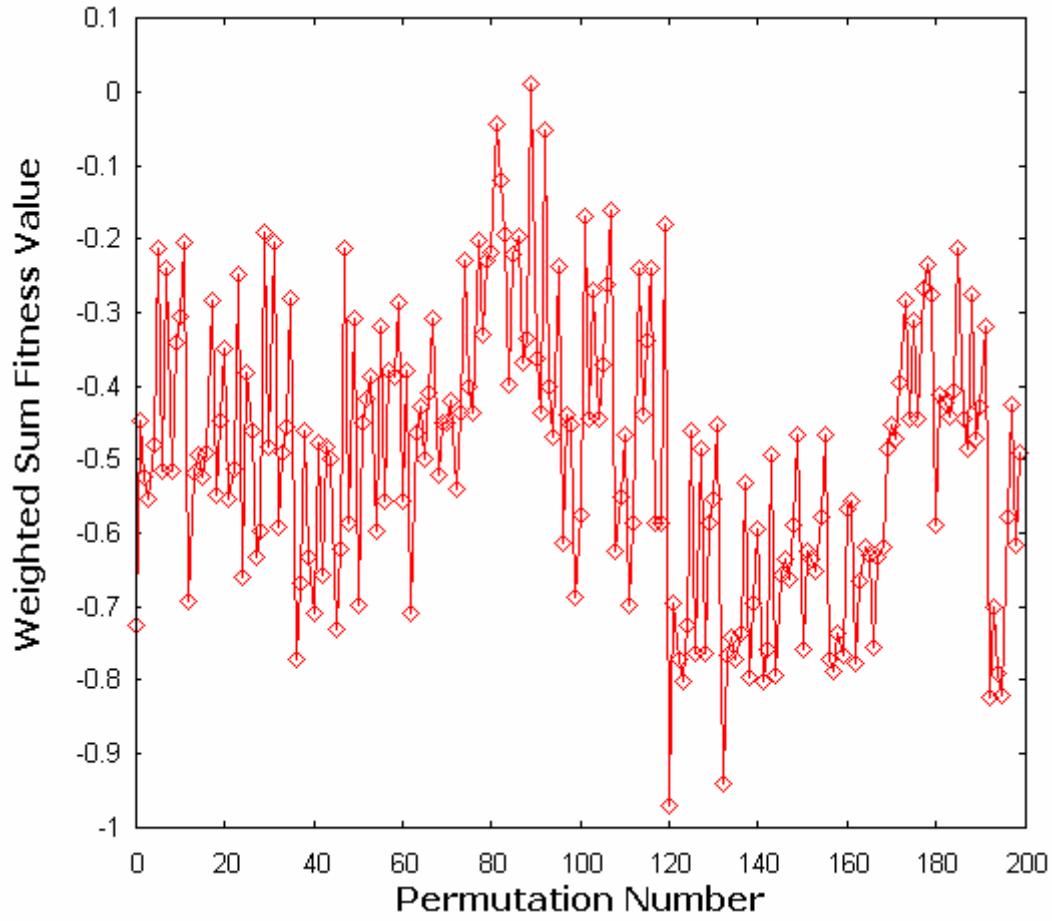


Figure-35 *Part of Landscape of a Small Problem*

As it can be seen from Figure-35, exploration of such a landscape would be very difficult. If an early convergence is allowed, the probability of getting stuck on a local maximum is very high, because of the highly cluttered nature of the landscape. However, in order to reach the global maximum, exploration should be done as much as possible, by using very high crossover and

mutation rates. When such landscapes are encountered, it must be usual for a genetic algorithm to have an inclination towards random search, while still making use of its inherent advantages.

6.2.5 Improving Performance of Weighted Sum

According to the parameter configuration analysis, it was realized that the exploration-exploitation trade-off requires high elitism and high mutation and crossover rates. Therefore, the following parameter values and selection and crossover techniques are determined to be used on the problem:

Table-9 *Improved Parameter Configuration for Weighted Sum*

Parameter	Value
CROSSOVER RATE	1.0
MUTATION RATE	1.0
SELECTION METHOD	TOURNAMENT 20 (ELITISM 5 %)
CROSSOVER METHOD	CYCLE CROSSOVER (CX)

Again, population size is taken 100 and the algorithm is run 200 generations. According to this improved parameter configuration, the results demonstrated in Table-10 are obtained:

Table-10 *Numerical Results of Weighted Sum with Improved Parameter Configuration*

USED WEIGHT CONFIGURATION			***SOLUTION***		
HIERARCHICAL WEIGHT %	**TEAM WEIGHT %**	**WEIGHT TOTAL WEIGHT %**	**HIERARCHICAL VIOLATIONS**	**TEAM VIOLATIONS**	**WEIGHT TOTAL**
33.3	*33.3*	*33.3*	48	0	8283
50	*25*	*25*	37	0	7857
25	*50*	*25*	58	0	8350
25	*25*	*50*	67	0	8567
10	*45*	*45*	79	0	8715
45	*10*	*45*	42	34	8562
45	*45*	*10*	38	0	7498
40	*30*	*30*	44	4	8084
30	*40*	*30*	47	0	8267
30	*30*	*40*	60	0	8444
30	*35*	*35*	46	4	8436
35	*30*	*35*	50	0	8269
35	*35*	*30*	54	0	8299
100	*0*	*0*	30	167	4917
0	*100*	*0*	240	0	4769
0	*0*	*100*	231	183	9399

If these values are compared with the results of the previous configuration (Table-4), incredibly better results are obtained, as Figure-36 illustrates:

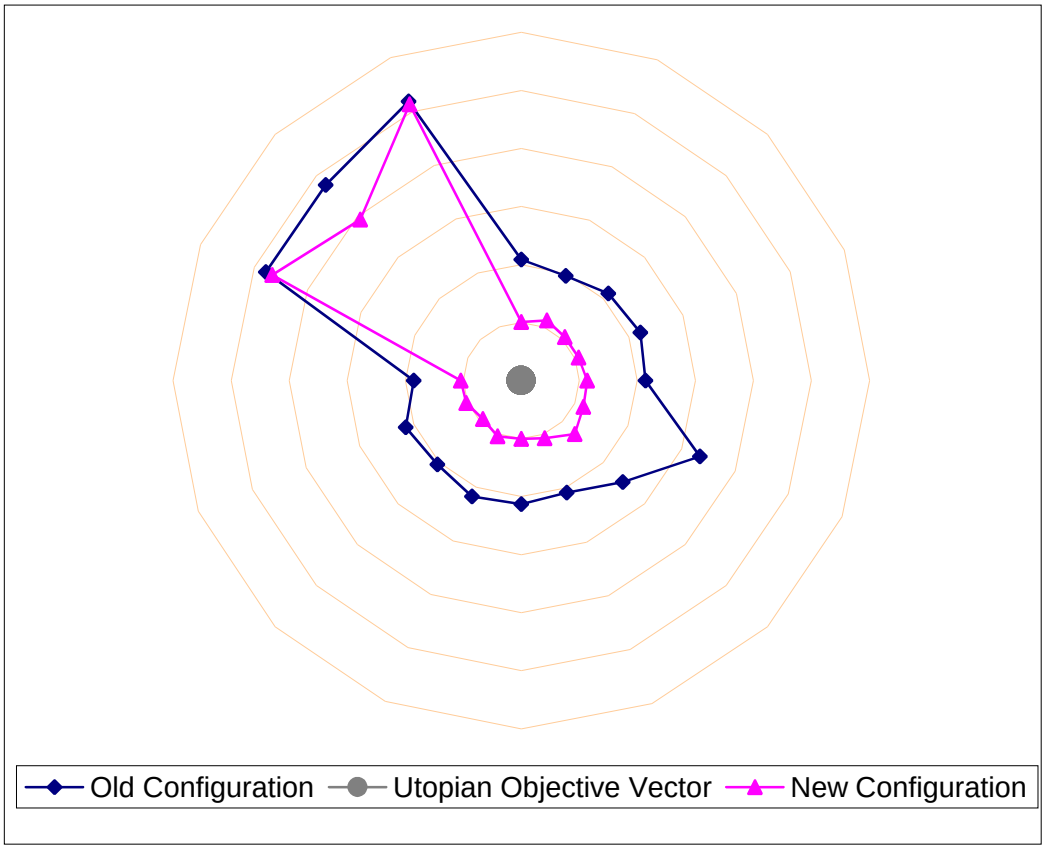


Figure-36 *Euclidian Distance with Improved Parameter Configuration*

6.3 Weighted-VEGA

In Weighted Sum, the crossover and mutation are directly applied to the current population. However, in VEGA, a mating pool is created explicitly before crossover and mutation. The mating pool concept that is found in VEGA was implemented in the same way by selecting individuals according to each objective in turn and inserting them into the mating pool. By doing this, it was expected to keep also the other objectives in control when assigning extreme weight configurations that will concentrate on a single objective but ignore the others. After applying this scheme,

considerably better results were obtained, not only in extreme weight configurations, but also in the others that care about all objectives:

Table-11 *Numerical Results of Weighted-VEGA*

USED WEIGHT CONFIGURATION			***SOLUTION***		
HIERARCHICAL WEIGHT %	**TEAM WEIGHT %**	**WEIGHT TOTAL WEIGHT %**	**HIERARCHICAL VIOLATIONS**	**TEAM VIOLATIONS**	**WEIGHT TOTAL**
33.3	*33.3*	*33.3*	46	0	8513
50	*25*	*25*	35	0	8007
25	*50*	*25*	48	0	8652
25	*25*	*50*	57	0	8719
10	*45*	*45*	68	0	8744
45	*10*	*45*	43	8	8678
45	*45*	*10*	37	0	7800
40	*30*	*30*	42	0	8343
30	*40*	*30*	42	0	8379
30	*30*	*40*	51	0	8529
30	*35*	*35*	53	4	8466
35	*30*	*35*	52	0	8419
35	*35*	*30*	47	0	8468
100	*0*	*0*	31	137	6568
0	*100*	*0*	156	0	5928
0	*0*	*100*	266	172	9474

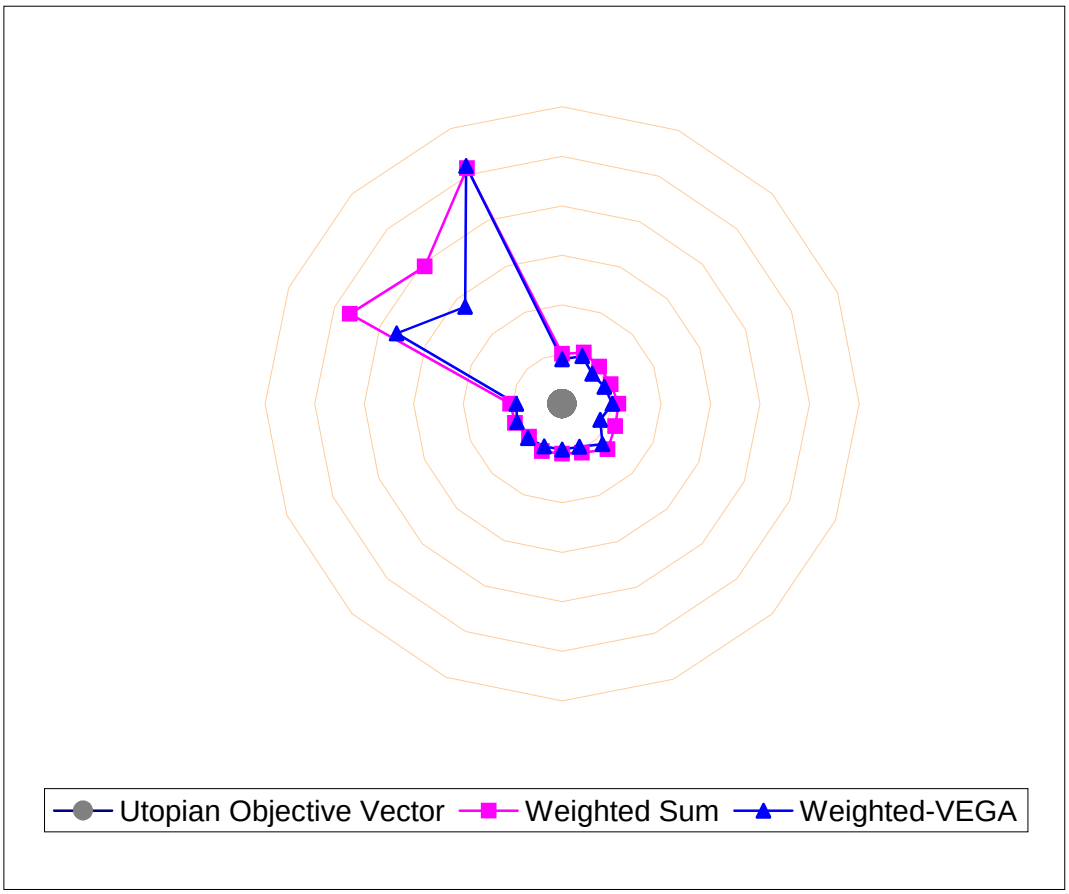


Figure-37 *Euclidian Distance of Weighted-VEGA Results*

As Figure-37 illustrates, by populating the mating pool with individuals by selecting them using each objective in turn, as it is also the case in VEGA, considerably better overall results can be obtained. When populating the mating pool, again elitism was preferred. Thus, elitism is being applied in two steps: once in populating the mating pool, and once in selecting the individuals for crossover. There is no doubt that elitism has an important contribution in the success of Weighted Sum.

CHAPTER 7

CONCLUSION

In this work, three different multi-objective optimization techniques, each of which represent a different class of multi-objective evolutionary algorithms are employed on multi-objective personnel assignment problem. According to the outcome of this analysis, the classical weighted sum method outperformed the other two in both diversity of solutions and closeness of solutions to the pareto-optimal front. Although SPEA is shown to be more successful in a set of problems in **[Zitzler et al. 2000]**, it did not turn out to be the same in this thesis work. The main reason for this may be the specific landscape of the problem. As it can also be seen from Figure-35, the landscape of multi-objective personnel assignment problem is highly cluttered. Moreover, its pareto-optimal front, which is highly non-convex, consists of a narrow set of solutions, which are dispersed non-uniformly and discontinuously (Figure-21). Under these extreme circumstances, multi-objective evolutionary algorithms may not be expected to return good results. On the contrary, it would be more logical to concentrate on a single solution close to the utopian objective vector, instead of trying to find out a set of non-dominated solutions. According to the results obtained in this work, Weighted Sum returns such a solution at each run, which generally dominates a considerable number of solutions that are found by the other methods.

Because of the extreme landscape conditions, Weighted Sum favors high mutation and crossover rates, in order to widely explore the landscape. However, it also tries to set a balance between exploration and exploitation by favoring the best solutions, making use of the elitism concept. Employing a parameter configuration in this direction, incredibly good results can be obtained, compared to the common GA parameter configurations.

When the concept of creation of the mating pool that is employed in VEGA by selecting individuals according to each objective in turn is adapted to Weighted Sum, even better results can be obtained. This is because, by introducing this scheme, two steps of elitism is introduced. Moreover, extreme weight configurations which favor a single objective and ignore others could be better controlled by this extension of Weighted Sum.

REFERENCES

Bodenhofer, U. (**2004**). *Genetic Algorithms: Theory and Applications Lecture Notes*. Third Edition. Johannes Kepler Universität Linz.

Caryl, M. (**1997**). *MUTANTS: A generic genetic algorithm toolkit for Ada 95*. http://www.permutationcity.co.uk/projects/mutants/ (last accessed on: 05.12.2005)

Cimen, Z. (**2001**). *A Multi-Objective Decision Support Model for the Turkish Armed Forces Personnel Assignment System*. Master's Thesis. AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH.

Cleveland, W. S. (**1994**). *Elements of Graphing Data*. Murray Hill, NJ: AT & T Bell Laboratories.

Coello, A. C. (**1999**). *A comprehensive survey of evolutionary-based multiobjective optimization techniques*. Knowledge and Information Systems, vol. 1, no. 3, pp. 269-308.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (**2001**). Introduction to Algorithms. M.I.T. Press, Cambridge, Massachusetts, U.S.A

Darwin, C. R. (**1859**) *On the Origin of Species by means of Natural Selection and The Descent of Man and Selection in Relation to Sex*, third ed., vol. 49 of Great Books of theWesternWorld, Editor in chief: M. J. Adler. Robert P. Gwinn, Chicago, IL, 1991. First edition John Murray, London.

Deb, K. (**2001**). *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons Publishing.

Davis, L. (**1985**). *Applying Adaptive Algorithms to Epistatic Domains*. Proceedings of the International Joint Conference on Artificial Intelligence, pp. 162-164.

Dinc, K., and Oguztuzun, H. (**1998**). *Personnel assignment for Turkish Land Forces* (in Turkish). In Bilisim'98, TBD 15th National Informatics Congress, Istanbul, pp. 303-307.

Edmonds J. (**1965**). *Maximum matching and a polyhedron with (0,1) vertices*. J. Res. Nat. Bur. Standards Sect. B, Vol. 8, pp. 125-130.

Ehrgott, M., Xavier, G. (**2000**). *An Annotated Bibliography of Multiobjective Combinatorial Optimization*. Technical Report 62/2000, Fachbereich Mathematik, Universitat Kaiserslautern, Kaiserslautern, Germany.

Eiben, A. E., Schippers, C. A. (**1998**). *On evolutionary exploration and exploitation*. Fundamentae Informaticae, 35:35-50.

Fonseca, C. M., Fleming, P. J. (**1995**). *An overview of evolutionary algorithms in multi-objective optimization*. Evolutionary Computing, Vol. 3, No. 1, pp. 1-16.

Fourman, M. P. (**1985**). *Compaction of symbolic layout using genetic algorithms*. Proc. Int. Conf. on Genetic Algorithms and Their Applications, Pittsburg, PA, pp. 141-153.

Garey, M.R., Johnson, D.S., Tarjan, R.E., and Yannakakis, M. (**1983**). *Scheduling opposing forests*. SIAM J. on Algebraic and Discrete Methods, Vol. 4, No. 1, pp. 72-93.

Galil, Z. (**1986**). *Efficient algorithms for finding maximum matching in graphs*. ACM Comp. Surv., Vol. 18, pp. 23-38.

Geoffrion, A. M., Dyer, J. S., and Feinberg, A. (**1972**). *An interactive approach for multi-criterion optimization with an application to the operation of an academic department*. Management Science 19 (4), 357-368.

Gen, M., Ida, K., and Li, Y. (**1995**). *Solving bicriteria solid transportation problem with fuzzy numbers by genetic algorithm*. International Journal of Computers and Industrial Engineering, 29:537—543.

Ghosh A., Nath B. (**2004**). *Multi-objective rule mining using genetic algorithms*. Information Sciences, Vol. 163, pp. 123-133.

Goldberg, D. E. (**1989**). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing.

Goldberg, D. E., Linge, Jr., R. (**1985**). *Alleles, Loci and the TSP*. Proc. 1st Int. Conf. Genetic Algorithms and Their Applications, Hillsdale, NJ, pp. 154-159.

Holland, J. H. (**1975**). *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press.

Jakob, W., Gorges-Schleuter, M., and Blume, C. (**1992**). *Application of genetic algorithms to task planning and learning*. In R. Manner and B. Manderick, editors, Parallel Problem Solving from Nature PPSN'2, LNCS, pages 291--300, Amsterdam. North-Holland.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (**1983**). *Optimization by Simulated Annealing*. Science, 220, 671-680.

Kuhn, H.W. (**1955**). *The Hungarian method for the assignment problem*. Naval Research Logistics Quarterly, Vol. 2, pp. 83-97.

Kursawe, F. (**1992**). *Evolution strategies for vector optimization*. Proc. 10th Int. Conf. Multiple Criteria Decision Making, Hsinchu, Taiwan, pp. 187-193.

Larranga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., Dizdarevic, S. (**1999**) *Genetic algorithms for the Travelling Salesman Problem: A review of Representations and Operators*. Artificial Intelligence Review, 13, 1999, pp. 129-170.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B. (**1985**). *The Traveling Salesman Problem : A Guided Tour of Combinatorial Optimization*. John Wiley & Sons.

Liu, X., Begg, D., and Fishwick, R. J. (**1998**) *Genetic approach to optimal topology/controller design of adaptive structures*. International Journal for Numerical Methods in Engineering, 41:815—830.

Mehlhorn, K., Näher, S. (**1999**). *A Platform of Combinatorial and Geometric Computing LEDA*. Cambridge University Press.

Meisel, W. L. (**1973**). *Tradeoff decision in multiple criteria decision making*. In J. L. Cochrane and M. Zeleny (Eds), Multiple Criteria Decision Making, pp. 461-476. Columbia, SC: University of South Carolina Press.

Mitchell, M. (**1996**). *An Introduction to Genetic Algorithms*. MIT Press.

Munkres, J. (**1957**). *Algorithms for assignment and transportation problems*. J. Soc. Indust. Appl. Math., Vol. 5, pp. 32-38.

Oguztuzun, H., Toroslu, I. H., Iskender, C. (**1999**). *On the assignment problem under ordering constraint*. ISCIS'99, Kusadasi, pp. 273-279.

Oliver, I. M., Smith, D. J., and Holland, J. R. C. (**1987**). *A Study of Permutation Crossover Operators on the Traveling Salesman Problem*. Proceedings of the 2nd International Conference on Genetic Algorithms, 224-230.

Pearl, J. (**1984**). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, Massachusetts.

Rao, S.S. (**1984**). *Optimization: Theory and Applications*. New York: Wiley.

Ritzel B. J., Wayland E. J., Ranjithan S. (**1994**). *Using genetic algorithms to solve a multiple objective groundwater pollution containment problem*. Water Resources Research, 30, 5, 1589-1603.

Rubenstein-Montano, B., Malaga, R. A. (**2002**). *A weighted sum genetic algorithm to support multiple-party multiple-objective negotiations*. IEEE Transactions on Evolutionary Computation, Vol. 6, No. 4, pp. 366 – 377.

Russell, S. J., Norvig, P. (**1995**). *Artificial intelligence: a modern approach.* Second edition, Prentice-Hall.

Schaffer, J. D. (**1985**). *Multiple objective optimization with vector evaluated genetic algorithms*. Proc. Int. Conf. on Genetic Algorithms and Their Applications, Pittsburg, PA, pp. 93-100.

Surry, P. D., Radcliffe, N. J., and Boyd, I. D. (**1995**). *A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks : The COMOGA Method*. In Terence C. Fogarty, editor, Evolutionary Computing. AISB Workshop. Selected Papers, Lecture Notes in Computer Science, pages 166-180. Springer-Verlag, Sheeld, U.K.

Toroslu, I. H. (**2003**). *Personnel assignment problem with hierarchical ordering constraints*. Computers and Industrial Engineering, Vol. 45, pp. 493-510.

Ucoluk, G. (**2002**). *Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation*. Intelligent Automation and Soft Computing, 3(8), TSI Press, 2002.

Wall, M. (**1996**). *GAlib: A C++ Library of Genetic Algorithm Components version 2.4 Documentation Revision B*. http://lancet.mit.edu/ga/dist/galibdoc.pdf (last accessed on: 05.12.2005)

Zitzler, E. (**1999**). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications.* PhD Thesis, Swiss Federal Institute of Technology (ETH) Zurich. TIK-Schriftenreihe Nr. 30, Diss ETH No. 13398.

Zitzler, E., Deb K., Thiele L. (**2000**). *Comparison of Multiobjective Evolutionary Algorithms: Empirical Results*. Evolutionary Computation, Vol. 8, No. 2, pp.173-195.

Zitzler, E., Thiele, L. (**1998**). *An Evolutionary Algorithm for Multiobjective optimization: The stength pareto approach*. in Tech. Rep. 43, Computer Engineering and Communication Network Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Gloriastrasse, CH-8092 Zurich.

Zitzler, E., Thiele, L. (**1999**). *Multi-objective evolutionary algorithms: A comparative case study and strength Pareto approach*. IEEE Trans. on Evolutionary Computing, Vol. 3, No. 4, pp. 257-271.