# A HYBRID MULTI-OBJECTIVE GENETIC ALGORITHM FOR BANDWIDTH MULTI-COLORING PROBLEM

by

İsmail Uğur Bayındır

Submitted to the Institute of Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Computer Engineering

Yeditepe University
2014

A HYBRID MULTI-OBJECTIVE GENETIC ALGORITHM FOR BANDWIDTH MULTI-COLORING PROBLEM

APPROVED BY:

Assoc. Prof. Dr. Emin Erkan KORKMAZ: .....
(Supervisor)

Assoc. Prof. Dr. Nafiz ARICA: .....

Assist. Prof. Dr. Dionysis GOULARAS: .....

DATE OF APPROVAL: ..../..../2014

# ACKNOWLEDGEMENTS

I am especially thankful to my supervisor, Assoc. Prof. Dr. Emin Erkan Korkmaz, whose support and guidance from the initial to the final level enabled me to accomplish this research.

I would also like to thank Assoc. Prof. Dr. Nafiz Arıca and Assist. Prof. Dr. Dionysis Goularas for their valuable advises to make this thesis better.

# ABSTRACT


## A HYBRID MULTI-OBJECTIVE GENETIC ALGORITHM FOR BANDWIDTH MULTI-COLORING PROBLEM

Genetic Algorithms (GAs) have been successfully applied on different kinds of problems. Multi-objective Genetic Algorithms (MOGAs) are capable of improving different objectives in a parallel manner. Various applications of MOGAs exist for combinatorial optimization problems. However, the MOGA approach yields a limited success rate especially on grouping problems. The crossover operation, one of the reproduction methods in GAs, is the main reason for the low performance. The crossover operation is quite destructive in grouping problems and it is difficult to produce successful offspring with this operator in this domain. In this study, a novel method that can increase the success rate of crossover operation is proposed for grouping problems. The method is a hybridization of MOGA with Artificial Neural Networks (ANNs), where ANNs guide the crossover process in the genetic search. The bandwidth multicoloring problem where standard MOGA yields limited performance has been used as the testbed for the method. The problem is solved using a multi-objective framework that minimizes bandwidth as well as conflict number in a parallel fashion. It has been observed that the crossover operation guided by the trained ANN improves the possibility of producing high fit offspring and the quality of the overall solution obtained at the end of MOGA runs.

# ÖZET


## ÇİZGEYİ KÜMELİ BOYAMA PROBLEMİ İÇİN KULLANILAN ÇOK HEDEFLİ HİBRİT GENETİK ALGORİTMA

Genetik Algoritmalar (GAs) çeşitli problemler üzerinde başarıyla uygulanmıştır. Çok hedefli Genetik Algoritmalar (ÇHGAs) birbirinden farklı hedefleri paralel olarak iyileştirebilmektedir. Kombinatoryal optimizasyon problemleri için çeşitli ÇHGA uyguları vardır. Ancak, ÇHGA yaklaşımı özellikle gruplama problemi üzerinde sınırlı seviyede başarı oranına sahiptir. Bu düşük başarı oranından çoğalma yöntemlerinden biri olan çaprazlama operatörü sorumludur. Çaprazlama operatörü, gruplama problemi üzerinde yıkıcıdır ve bu tür problemler üzerinde çaprazlama operatörü kullanarak başarılı yeni bireyler üretilmesi zordur. Bu çalışmada, çaprazlama operatörünün gruplama problemleri üzerindeki başarı oranını arttıran yenilikçi bir metot sunulmuştur. Metot ÇHGA'nın Yapay Sinir Ağları (YSA) ile melezlenmesinden oluşmaktadır; YSA çaprazlama operasyonuna genetik arama işlemi sırasında yol göstermektedir. Sunulan metot, standart ÇHGA'ların sınırlı başarı elde ettiği Çizgeyi Kümeli Boyama problemi üzerinde test edilmiştir. Problem, bant genişliği ve çakışma sayısının aynı anda azaltılmaya çalışıldığı bir çok-hedefli gerçekleme kullanılarak çözülmüştür. Yapılan testler sonucunda, YGS tarafından yönlendirilmiş olan çaprazlama operasyonunun başarılı birey üretme olasılığını arttırdığı ve elde edilen genel çözümlerin kalitesinin de yükseldiği görülmüştür.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS / ABBREVIATIONS

| | |
|---|---|
| ANN | Artificial Neural Network |
| ALS | Ant Local Search |
| BCP | Bandwidth Coloring Problem |
| BMCP | Bandwidth Multi Coloring Problem |
| EA | Evolutionary Approach |
| ENZO | Evolutiver Netzwerk-Optimierer |
| FAP | Frequency Assignment Problem |
| GTS | Generic Tabu Search |
| GA | Genetic Algorithm |
| GCP | Graph Coloring Problem |
| HC | Hill Climbing |
| HEA | Hybrid Evolutionary Algorithm |
| IG | Iterated Greedy |
| LLE | Linear Linkage Encoding |
| LS | Local Search |
| MA | Memetic Algorithm |
| M-PEAS | Memetic-PAES |
| MCP | Multi Coloring Problem |
| MOGA | Multi Objective Genetic Algorithm |
| MOOP | Multi Objective Optimization Problem |
| PSA | Parallel Simulated Annealing |
| PAES | Pareto Archived Evolution Strategy |
| SWO | Squeaky Wheel Optimization |
| SA | Simulated Annealing |
| SPEA | Strength Pareto Evolutionary Algorithm |
| TS | Tabu Search |

# 1. INTRODUCTION

Genetic algorithms (GA) have been widely used to solve various optimization problems. GAs keep a population of individuals named as chromosomes. The potential solutions have to be encoded as chromosome structures in GAs. Each chromosome has a fitness value denoting how close it is to the global solution. Genetic operators are utilized to generate the offspring by using the parents chosen from the population using a selection method. The search is carried out until the optimum solution appears in the population or until a predefined threshold value is reached for breeding new populations.

Reproduction process consists of crossover and mutation operations. Chromosomes with high fitness value have more chance to be selected to create the offspring. Crossover and mutation operators help carrying genetic materials to the next generations. The search process is mainly carried out with the help of the crossover and mutation operators. The crossover operation is expected to create more fit offspring compared to the parents and mutation is utilized in order to avoid converging to local optimum.

Artificial Neural Networks (ANNs) are computational models that have been used for machine learning and pattern recognition tasks successfully. ANNs have three elements; network properties, vertex properties and system dynamics. The network properties consist of network topology, type of connections, order of connections and weight range. The vertex properties are activation range and activation function. The system dynamics include weight initialization scheme, activation-calculating formula and the learning rule. If sufficient number of layers is utilized in an ANN, it is expected to approximate any arbitrary continuous function [1].

Graph Coloring Problem (GCP) is a well known NP-Hard combinatorial optimization problem. The aim in GCP is to assign a color to each vertex such that there will no adjacent vertices with the same color in the coloring schema formed. If two vertices are connected with an edge, then they are named as adjacent. Certainly, the optimum solution of the problem is the coloring that would be obtained by using minimum number of colors.

Different real-world problems such as time scheduling and resource assignment can be reduced to graph coloring problem.

This thesis focuses on Bandwidth Multi-Coloring Problem (BMCP), which is a generalization of GCP. There is a color separation within the vertices as well as adjacent vertices and a vertex can have multiple colors. In BMCP, each vertex and edge has a weight where the weight of a vertex denotes how many colors should be assigned to that vertex and the edge weight denotes the minimum color separation that should exist between the vertices sharing this edge. BMCP can be reduced to GCP by setting all vertex and edge weights to one.

In BMCP, there are multiple objectives to be optimized. The number of colors used in a coloring scheme is named as the bandwidth in BMCP. The bandwidth size forms the first objective to be minimized. Certainly, a coloring scheme might have conflicts which are the colors violating the constraints defined in the previous paragraph. Minimizing the number of conflicting colors is the second objective of the problem. Due to multi-objective nature of the problem, it is not possible to determine the superior chromosome, unless one dominates the other in both of the objectives. A MOGA approach similar to [2, 3] is utilized in this study. The main advantage of MOGA approach is the fact that it is possible to obtain an approximation of the entire Pareto front in a single algorithm run. Hence, the optimal bandwidth size does not need to be determined beforehand unlike other approaches in the literature.

The frequency assignment problem (FAP) can be modeled as BMCP. In the last decade, wireless services like digital cellular phone networks have developed rapidly. This increased the need for the important resource which is the frequencies in the radio spectrum. Frequencies within a wireless communication network can be reused. However, this may decrease the quality of communication links. The frequency assignment problem tries to balance the reuse of frequencies and the loss of quality in the network [4].

MOGAs have been used for solving combinatorial optimization problems in the literature. Evolutionary approaches have a limited success especially on grouping problems. Crossover is the main operator that provides convergence in evolutionary methods.

However, the operation is quite destructive in grouping problems and it is difficult to produce high fit offspring with this operator in this domain. Therefore the genetic algorithm applications provided for grouping problems are usually hybrid methods that include hill climbers or search methods like Tabu search [5, 6]. In this study, a new crossover method is proposed for grouping problems. It is tested with the MOGA framework on BMCP in this study, but it can be generalized to other evolutionary approaches and other problems. The method is based on guiding the crossover process by Artificial Neural Networks. Standard MOGA yields limited performance on BMCP. Therefore this problem has been selected as the testbed for the method.

Group crossover is widely used for grouping problems [7]. GCP and its generalizations are also grouping problems. GCP can be considered as partitioning the vertices of a graph into groups such that the vertices in the same group can be colored with the same color. In group crossover operation, after the parents are chosen, the groups that exist in the parents are determined. Then these groups are transferred to the offspring using a strategy. For instance in [7] the largest groups are transferred one by one to the offspring by switching from one parent to another. However, this operation has a low performance on BMCP due to the characteristic of the problem. In this study, uniform crossover operation is utilized on BMCP. However, the strategy used to select the genes in parents is guided by a trained ANN. In the initial phase of the MOGA search, the standard uniform crossover is utilized where the colors that will be transferred to the offspring are selected randomly. However, the crossover operations carried out and the resulting offspring created are collected to form a training data. After a certain amount of generations, this data is used to train an ANN. Then whenever the crossover operation is to be performed, different transfer scenarios are formed and the trained ANN is used to select the best alternative. Then the crossover operation is performed accordingly.

It has been observed that the crossover operation guided by the trained ANN improves the possibility of producing high fit offspring. Certainly, the quality of the overall solution obtained at the end of a MOGA run also increases when better offspring are obtained as the result of the crossover operation.

Layout of the thesis is as follows: the formal problem definitions are presented and an overview of the solution methodologies for GCP and BMCP are given in the next chapter. The hybrid use of ANNs and evolutionary approaches is also presented in the next chapter. Overview of the proposed framework is given in Chapter 3. The detailed explanations of utilized methods and operators are also given in the Chapter 3.

The benchmark problem set utilized in this study is explained in the Chapter 4. Experimental results and comparison with the other state-of-the-art algorithms are also provided in this chapter. In the last chapter, the conclusion and future works are presented.

# 2. BACKGROUND

## 2.1. MULTI OBJECTIVE GENETIC ALGORITHMS

Most of the optimization problems involve more than one objective to be optimized in real world application. The objectives in the most of real life problems are often conflicting. One solid solution would not satisfy both objective functions and the optimal solution of one objective will not necessary be the best solution for the problem's other objective. Therefore, a set of solutions is required to represent the optimal solutions for all objectives in multi objective optimization problems.

The characteristic of evolutionary methods which use a population based solutions is well suited for multi objective optimization problems (MOOPs). The approaches that are used to solve MOOP aims to find a set of non-dominated solutions. These approaches can generate approximation of a pareto front, which is a set of non-dominated solution, in each generation. This aspect makes these approaches suitable for the MOOP. Requirement of little prior knowledge from the problem, less vulnerability to shape and continuity of pareto-front, easy implementation, robustness and the ability to be carried out in parallel are some of the advantages of evolutionary algorithms.

GAs which being a population based approach are well suited for solving MOOPs. GAs are inspired by the evolutionist theory explaining the origin of species. In nature, the strong individuals have greater opportunity to pass their genes to future generation via reproduction than week and low-fit individuals. A generic single objective GA can be modified to find a set of multiple non-dominated solutions in a single run. GAs are able to simultaneously search different regions of a solution space. This allows GAs to find a diverse set of solutions which is required for difficult MOOPs. The crossover operator of GA may exploit critical points of good solutions with respect to different objectives to create new non-dominated individuals in unexplored regions of the pareto front. These features make GA the most popular heuristic approach to MOOPs.

## 2.2. PROBLEM DEFINITION

The definition of the classical GCP is as follows. Given an undirected graph that consists of a set of vertices *V* and a set of edges *E*, the aim is to find a minimum color number *k* and a mapping *R* of these *k* colors to each *V* in the graph such that two vertices that share an edge cannot have the same color. Hence, each vertex in the graph is assigned a single color and the adjacent vertices have different colors.

In bandwidth coloring problem (BCP), again an undirected graph consisting of set of vertices *V* and a set of edges *E* is colored. However the edges are assigned edge weights *d(i,j)*, where *i* and *j* are two vertices. The aim is again to find a minimum color number *k* and a mapping *R* between the colors and vertices in a such that two vertices *i*, *j* that share an edge should have a color difference greater than or equal to the edge weight *d(i,j)* assigned to this edge. Hence, the colors have to be indexed in BCP and vertices that share an edge should be assigned to two colors that have a index difference that satisfy the edge weight constraint.

The Multi Coloring Problem (MCP) is another generalization of the GCP. In this problem, each vertex can be assigned multiple colors. Color count is determined by the vertex weight. For example, if a vertex has weight four, then four distinct colors have to be assigned to this vertex. Adjacent vertices still cannot share the same color.

BMCP is a combination of BCP and MCP. It is the most complex version of the GCP generalizations. The graph can contain self loops. Formally, given that a graph *G(V,E)* which has vertex weights *k(i)* for all individual $i \in V$, and edge weights *d(i,j)* for$(i,j) \in E$, intent is to determine a minimum *k* and subsets $S(i) \subset \{1,\dots,k\}$ for each $i \in V$, in a way that $|S(i)| = k(i)$ for each $i \in V$ and also $S(i) \cap S(j) = \emptyset$ and where, for each $p \in S(i)$ and $q \in S(j)$, $|p-q| \geq d(i,j)$ for each $(i,j) \in E$. As an example consider the graph in Figure 2.1. In this figure, $d(1,1) = 1$ and $k(1) = 4$. Hence, four colors should be assigned to vertex one and the difference between the colors assigned should be at least three.

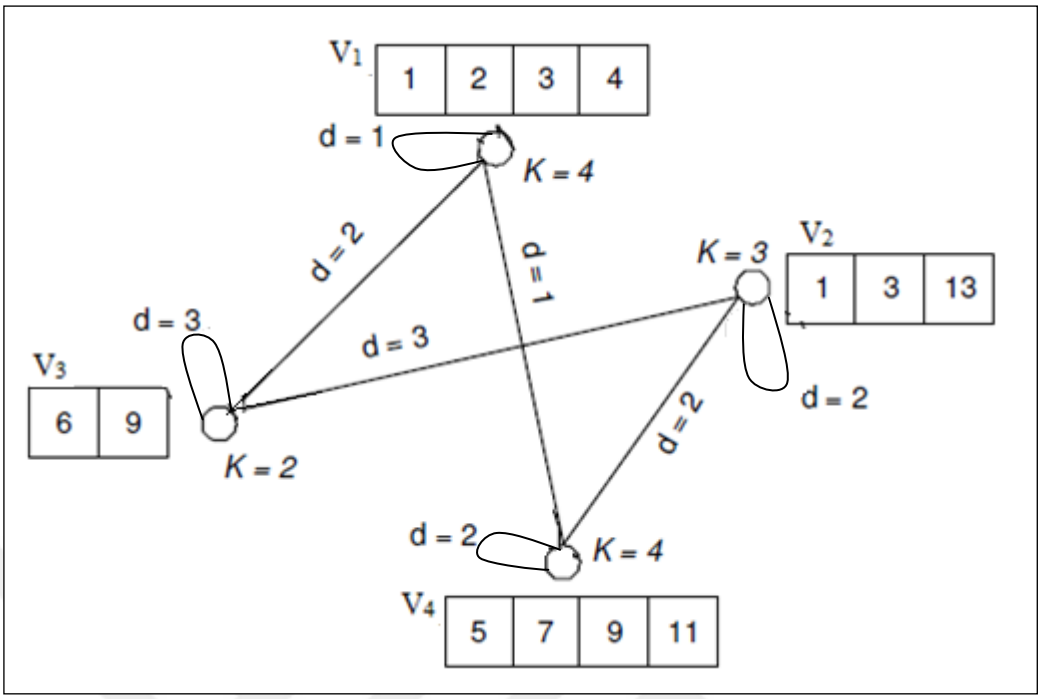


Figure 2.1. An example of BMCP graph

## 2.3. EXAMPLE STUDIES FOR GCP AND BMCP

GAs have been utilized for grouping problems by researchers. However, the performance of GAs fall behind compared to the other methods used to solve grouping problems. The genetic operators might be destructive on the individuals when GAs are applied. This is usually the main reason for the low performance of GAs compared to the other methods. Representation is a critical issue in order to prevent the damage of genetic operators in the GA search.

Linear Linkage Encoding (LLE) is proposed for GAs as an encoding scheme for grouping problems and it has been used with a multi-objective GA in [8]. When LLE is used with standard reproduction operator such as one-point crossover, the convergence is not acceptable due to the bias produced in the search process. In [8], a new crossover operator is proposed for LLE. The operator is introduced to remove the bias caused by one-point crossover. All potential offspring have equal probability to be produced by the new crossover operator.

The elements of a single group can be spread along the chromosome in a grouping problem. Therefore one-point crossover is destructive when the building blocks are separated from each other. The ordering in the linear structure of LLE prevents one-point crossover to exploit all regions of the search space. Some partitions cannot be produced by using one-point crossover. Uniform crossover does not cause the same bias, because the genetic material to be passed to the offspring is chosen randomly from the parents. However, in some cases this process can introduce random perturbations on the individuals which are not an acceptable case. The crossover proposed in [9] is named as group-crossover. This operator aims to remove the disadvantages of one-point and uniform crossover operations.

In Figure 2.2, an example chromosome represented in LLE is given. In this representation each cluster is represented as a linked list of objects. A different gene is reserved for each object. The value of a gene denotes the id of the next object in the same group. Two objects are in the same group, if either one can be reached from the other one using the links.

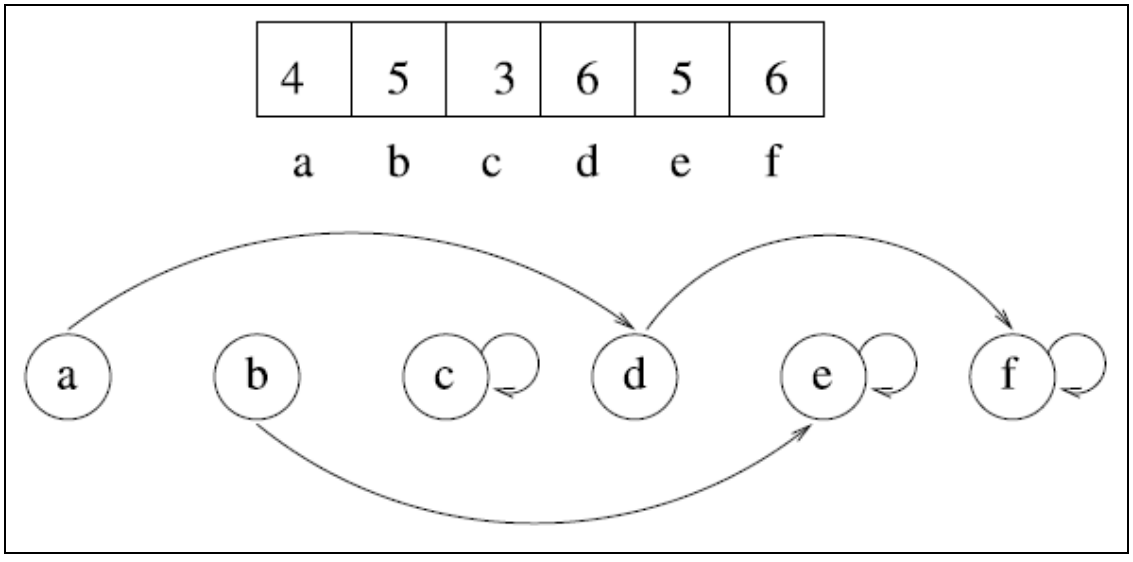


Figure 2.2. An example chromosome represented in LLE

Each group in LLE has a vertex in the chromosome that is linked to itself. These vertices are named as ending vertices. There ending vertices are focused in the group-crossover

operator. The crossover operator ignores the ordering of the elements in a group. Genetic materials are passed to the offspring from the parents based on these ending vertices.

If both $P_1[i]$ and $P_2[i]$, $i$ is the vertex number, are ending vertices; then $O[i]$ is set as an ending vertex. If only one of the parents is ending vertex, $O[i]$ is set as an ending vertex with probability 0,5. If $O[i]$ is not set as an ending vertex by the previous item, then the ending vertex of the groups that the $i^{th}$ element belongs to both parents are determined. If one of these ending vertex is transferred previously to the offspring, then $O[i]$ is linked to this ending vertex. If both of them are transferred, then $O[i]$ is randomly linked to one of them. If none of the ending vertices are transferred, then $O[i]$ is linked to the element that $i^{th}$ element is linked in one of the parents randomly.

Apart from the representation issue, researchers have focused on different methods to solve graph coloring problem and its generalizations. For instance, Tabu Search (TS) is utilized to solve graph coloring, T-coloring and set T-coloring problems in the literature. In [9], a generic TS is presented for three-coloring problem. Proposed algorithm integrates important features such as greed initialization, solution re-generation, dynamic tabu tenure, incremental evaluation of solutions and constraint handling techniques. A Generic Tabu Search (GTS) is presented in the study. The proposed GTS algorithm is consisting of three parts: Greedy construction of initial coloring, configuration re-generation and searching for proper coloring are the three algorithms used in the method. A Dsatur-based greedy algorithm [10] is used in the greedy construction of initial coloring. This greedy approach is fast and provides a good initial configuration. Getting fast and good initial configuration plays crucial role for the convergence. In the configuration re-generation part, the aim is to produce a $k-1$ coloring, $k$ being the maximum color value, with a minimum conflict number. This is obtained by coloring the vertices that are assigned color k with a new color in the range $[1, \dots, k-1]$. While coloring the vertices with a new color, conflict number is tried to be kept as low as possible. When searching for proper coloring, the tabu algorithm takes an improper coloring, (a coloring that has conflicts) and tries to remove all the conflicts by using the given $k^{th}$ color. If the algorithm finds a proper coloring by using the $k^{th}$ color, it proceeds to re-generation part to produce a new improper coloring with $k-1$ colors again.

Simulated Annealing (SA) is another method that has used to solve GCP. SA is a generic probabilistic metaheuristic for the global optimization problem. The technique of controlled cooling of a material to increase the size of its crystals and reduce its defects, annealing in metallurgy, is the inspiration of the SA method in optimization. At each step, the SA heuristic considers some neighboring state $s'$ of the current state $s$. Then, it probabilistically decides moving the system to state $s'$ or staying in state $s$.

An application of Parallel Simulated Annealing (PSA) is described in [11] to solve GCP. Proposed PSA algorithm utilizes multiple processors that are working at the same time on individual chains to find a solution at a fixed time. Then the routine minimizes the cost function by storing the best solution. The master-slave model has been used to provide coordination in the algorithm. Collection of states, choosing the next state and distributing it among the slave units is done by master processing unit.

There are also some hybrid methods that have been used to solve GCP and its generalizations in the literature. Iterated Greedy (IG) algorithm and Squeaky Wheel Optimization (SWO) have been adapted to develop a hybrid method in [14]. Given a permutation of vertices, the Greedy algorithm selects a vertex sequentially and assigns the next color that has not been used in any of its neighboring vertices. The IG algorithm is an extension of the greedy method. The IG algorithm uses a new permutation in each iteration. The new permutation is formed by ordering the vertices that have the same color in the previous coloring scheme [13]. Hence, the new coloring scheme will not have more colors compared to the previous coloring scheme. The three main components of SWO are Constructor, the Analyzer and the Prioritizer [14]. SWO also uses a permutation sorted in decreasing order of the vertex degrees; it is similar to IG method. The Constructor's job is to greedily construct a coloring scheme. Then the Analyzer assigns a blame to each vertex, if the vertex has a color beyond the target range. The target range is set as one less than number of colors in the current best solution. After the blames are set, the Prioritizer updates the previous permutation based on how much blame the vertices have. In the proposed method the Constructor component of SWO is replaced by a Hill-Climbing (HC) procedure. The HC method tries to improve the greedily generated solution by downhill moves. The HC method orders the vertices based on the color index they have. Again the approach is similar to the IG method.

TS heuristic is utilized in a SWO framework in [15]. In the proposed framework, solutions are modeled as sequences of vertices. A greedy algorithm is used to assign colors at the beginning. Then meta-heuristics are applied to find better solutions by adjusting these sequences. A vertex that has vertex weight k is split into k vertices to reduce the complexity in the representation model. The framework has two parts. The SWO method is used to adjust sequences in the first part. Then the best solution found by SWO is passed to TS for further improvement. TS strengthen the search procedure and avoid local optima in the search space. The greedy algorithm is used to determine the number of colors required to color the graph. Also a new method is used to calculate the blame values in the SWO module. Vertices that have a color index greater than the multiplication of blame rate and the maximum color index k are assigned a constant blame value. TS operates based on neighborhood moves. The exchange of two vertices in the solution sequence is considered as a neighborhood move. TS uses a tabu memory to prevent some unwanted reverse moves to happen.

Using hybrid evolutionary algorithms (HEA) on the GCP is also common in the literature. An algorithm that combines a highly specialized crossover operator and a well known tabu search algorithm is proposed in [7]. The algorithm performs a series of iterations called generations after the initial population is formed. In each generation, two parents are selected to apply crossover operation on them. The offspring produced by the crossover operation is improved by using a Local Search (LS) operator. Finally, the improved offspring is inserted back to the population. This process continues until a stopping criterion is met. This hybrid algorithm differs from a standard genetic algorithm, because the mutation operator in GA is replaced with a LS operator. Also a new crossover operator is proposed in this study. In crossover operation, firstly the group with maximum number of vertices is chosen from the selected parent and passed to the offspring. Then this group is removed from both parents. This process is repeated k times, k being number of groups that exist in both parents. After k steps, remaining unassigned vertices assigned to a random class in the offspring. The purpose of the LS operator is to improve the offspring produced by the crossover operator. Tabu search is utilized as the LS method. The algorithm chooses a vertex that is conflicting with another vertex and the conflicting vertex is moved to a different group to explore the neighbors of the current coloring scheme. Certainly, previously visited neighbors are prevented using a Tabu list.

E. Malaguti and p. Toth also proposed an evolutionary approach for BMCP in [16]. DSATUR [10] uses an ordering that maximizes a given score on the vertices. Then the algorithm chooses the first vertex from the ordered list and colors it. This ordered list is kept to color the vertices that are harder to color, at the beginning of the process. The number of distinctly colored adjacent vertices as well as the distance of color indexes within the vertex and its adjacent vertices determines the score of a vertex. Certainly, the vertices with a higher score are harder to color. A constructive heuristic is also used in this study. The population based TS utilized in the study uses partial solutions and tries to find perfect solution by coloring and uncoloring selected vertices. Distance Crossover is proposed in this research. Parents are selected randomly from the population pool. The important structures are transferred from the parents to the offspring based on color index distance between the vertices. First, the "tight distance" pairs from first parent are copied to the offspring. Then the "tight distance" pairs that do not cause conflict with the already colored vertices are copied to the offspring from the second parent. If a vertex cannot be colored without causing conflicts, it is uncolored. The vertices left uncolored are assigned colors by using a TS operator at the end of the process.

The researchers have been inspired by the natural events and processes. GAs form an example to this fact. Ants and their nature have inspired the researchers as well. An algorithm called Ant Local Search (ALS) is proposed for GCP in [17]. In the ant algorithm proposed in [17], each ant builds a solution step by step. At each step, an element is added to the current partial solution by an ant. The greedy force and the trails are the two ingredients of the ant algorithms. The greedy force can be defined as the short term profit for the considered ant. The information obtained from other ants is defined as the trails. In the ALS, each ant is considered as a local search to get competitive results. Each ant evolves the solution by performing random modification on it.

The IMPASSE class local search algorithms have given competitive results on many coloring benchmark problems. S. Prestwich has proposed an IMPASSE style LS algorithm in [19]. The IMPASSE class algorithms work on the coloration neighborhoods. The coloration neighborhood consists of the vertices that are colored and no adjacent vertices share the same color. Remaining uncolored vertices are called the impasse set. The LS algorithm tries to remove all vertices from the impasse set by coloring and uncoloring

some selected vertices. Vertices with large domain and small forward degree are selected for uncoloring. A vertex has large domain if it has many colors in the coloring scheme. The forward degree of a vertex is the number of uncolored adjacent vertices. While selecting a color index to color a vertex, remembering the colors used in the previous iterations increases the performance. The coloring rule flips between two modes; picking up a new color and picking up a color that is successfully used in the previous iterations. When a color id successfully assigned to a vertex, rule flips to the other method. This rule is used to minimize disruption in the coloration.

The frequency assignment problem (FAP) can be modeled as BMCP. Constraint handling methods have been used in evolutionary search. R. Dorne and J. Hao have developed constraint handling techniques for FAP in [20]. They aimed to minimize the electromagnetic interference due to frequency reuse and minimize the number of frequencies used in FAP in cellular radio networks. In this study EAs utilized without a crossover operator. Crossover operation is very destructive for the FAP. Selecting parents and the evaluation of offspring have been used as EA features. Mutation operator has been used as the single reproduction operator in the study.

A Memetic Algorithm (MA) has been proposed for multi objective optimization problems in [21]. The method employs the LS method used in the Pareto Archived Evolution Strategy (PAES) and it utilizes the LS together with a population based approach that includes reproduction operators. The memetic-PAES algorithm (M-PAES) is based on the local search multi objective algorithm used in [22]. It adds a population based approach and reproduction methods to local optima found by using the PAES approach. A finite sized archive consists of non-dominated solutions that are maintained by PEAS approach. The elements of this archive are the best solutions of the search process. The final solution is obtained from this archive and also the archive uses as a comparison set to determine the dominance rank of the new solutions. M-PEAS approach uses two different archives to manage these tasks; a global archive that maintains a finite set of the best solutions found so far and a local archive H that is used for the comparison procedure. These archives help the search process to converge quickly and steadily.

The Strength Pareto Evolutionary Algorithm (SPEA) [23] is a technique for finding or approximating Pareto-optimal set for multi objective optimization problems. Pareto approach guides the solutions in the search space for better convergence rate. SPEA2, an improved version of SPEA, is proposed by E. Zitzler et al. in [24]. A fine-grained fitness assignment strategy, a density estimation technique and an enhanced archive truncation method are used in SPEA2. These techniques are the main differences of SPEA2 compared to SPEA. Each individual is assigned to a fitness value based on how many individuals it dominates and it is dominated by. Each solution has a density estimation determined by the distance of k-th nearest neighbor of the solution in the close proximity. This neighbor density estimation technique allows a more precise guidance of the search process. A new archive truncation method is used to provide diversity in the pareto set. The method removes the solutions that are close to each other in the archive. The proposed framework is applied successfully to three instances of the knapsack problem and SPH-m which is a multi objective generalization of the Sphere Model.

## 2.4. ARTIFICIAL NEURAL NETWORKS (ANNS) AND EVOLUTIONARY APPROACHES

In computer science and related fields, Artificial Neural Networks (ANNs) are computational models inspired from the central nervous system. The method is applicable to machine learning as well as pattern recognition problems. These computational models are capable of learning from samples and making decisions [25].

The structure of the network determines whether one neuron may influence another. The extent of possible influence is specified by the weight assigned to each connection. It is a straightforward idea to use the Evolutionary Approach (EA) to assist neural network design and training. A global and very broad search process supplied by EAs can increase performance of the previous methods used in neural network design and training.

Setting the weights of a network can be seen as an optimization problem. The aim is to find a set of weights that minimizes the network’s error on the training phase. The search space is highly complex and usually contains many local minima. The most commonly used algorithm for the problem is the backpropagation method [26]. The algorithm often yields

poor result without problem specific parameter settings. EAs usually avoid local minima by running the search process on several regions of the search space simultaneously. This makes EAs suitable for optimization problems with many local minima. EAs only need a fitness evaluation function to carry out the search process and they are not restricted by the network topology. Due to these characteristics of the EAs, they have been used in neural network training in various studies in literature [27, 28, 29].

If number of neurons and number of connection between those neurons are small, the network might not be able to learn the desired input-output mapping. On the other hand, if large amount of neurons are used with highly connected network topology, inputs might be mapped to undesired outputs. The topology also influences speed and accuracy of the learning process in the network. EAs are also used to determine the structure of a neural network in [30]. As seen in the literature EAs are used to determine the structure of a neural network as well as to train the network separately. There are also some studies that EAs have been used to determine weights and structure of a neural network at the same time. [31, 32] are the examples to these cases.

The method proposed in [28] is one of the studies that use a genetic algorithm to train a feedforward neural network. The weights in the neural network are encoded as a list of real numbers. Evaluation function is a critical feature in GAs. First, the weights in chromosomes are initialized. Then, the network is run over the training set and the sum of the squared errors is set as the fitness value of the chromosome. While the population is initialized, the weights are uniformly distributed between -1.0 and 1.0. After the population initialization phase, the standard GA operators are applied on the chromosomes to get a new set of weights and the genetic search is carried out until the best set of the weights are determined.

The architecture of an ANN is also important for getting satisfactory results when the method is applied to practical problem domains. Different architectures can be chosen and trained in order to determine the architecture with best performance for the problem. But still there might be more suitable topologies not taken into consideration for the problem. A GA driven network generator that evolves ANN architecture is presented in [31]. The

framework is called as Evolutiver Netzwerk-Optimierer (ENZO). ENZO optimizes both the network topology and the connection weights simultaneously.

Network architecture is encoded into the chromosome structure. Every gene in a chromosome represents one connection in the network. The number of possible connections is fixed and GA is used to find the optimal topology. ENZO generates an initial population where each chromosome represents a different network. Each network has about $P1 \times 100$ of the total number of connections, $P_1$ being the connection density between zero and one. Each of the potential connections is established with the given probability $P_1$. These networks are trained, evaluated and sorted due to their fitness values. Then ENZO starts to create offspring using crossover and/or mutation. In the crossover procedure, if a connection is present in both parents, it is transferred to the offspring. If the connection is present in only one of the parents, it is transferred to the offspring by using a certain probability. Mutation operator changes the state of each potential connection by using again a given probability. After the reproduction procedures, the offspring is evaluated and inserted into the population according to its fitness value. Then the chromosome with the lowest fitness value is removed from the population. The framework is able to search the optimal topology in the search space that has high diversity. The quality of the learning process is increased due to GA's ability to find the optimum solution in the search space.

GAs are used to design and train ANNs, in all of the studies presented above. There is only one study where ANNs are used to enhance the search process in Gas. In [33], ANNs are used to determine parent pairs for the crossover operation. In this study, the convenient parent pairs that have the potential to produce high fit offspring are determined by a trained neural network. The structural properties of the chromosomes that are likely to produce high fit offspring are analyzed by an ANN. The ANN is expected to combine the parents in such a way that the fitness of the offspring would be high. Coherent building blocks of the chosen chromosomes are considered by the ANN while combining the parents. The training data is formed by the parents that are used in the crossover operator and the fitness value of the corresponding offspring. In the proposed framework, the first parent is chosen by using tournament selection. Then, the ANN is used to determine an appropriate mate for the first parent.

# 3. METHODOLOGY

## 3.1. OVERVIEW OF THE FRAMEWORK

In this thesis a Genetic Algorithm (GA) that is hybridized with a Local Search (LS) algorithm is proposed to solve the Bandwidth Multi Coloring Problem (BMCP). Bandwidth size and number of conflicts in the coloring scheme are both reduced in a parallel manner by the algorithm. The population is expected to converge as shown in Figure 3.1 in terms of both objective functions. MOGAs have not been utilized for BMCP before. The framework proposed is enhanced by a novel crossover operator where the process is guided by a trained ANN.

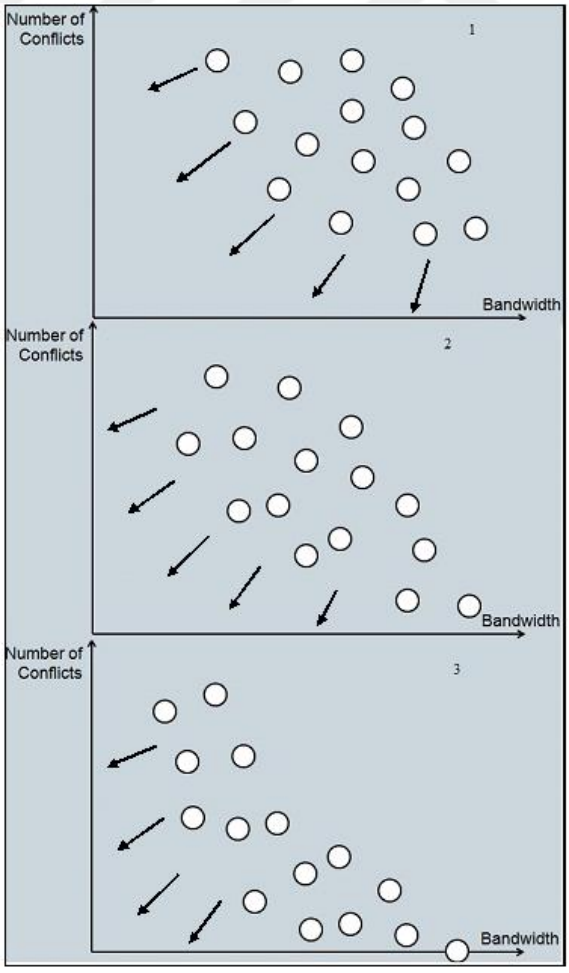


Figure 3.1. The convergence in the search process

Genetic operators are used to provide high variety chromosomes in the population. The chromosomes that are produced by the genetic operators are improved by using local search. Local search improves the convergence in the search process, since it covers the lack of fine tuning in GAs. Basic structure of the main algorithm utilized in this thesis can be seen in Algorithm 3.1.

Algorithm 3.1. Main structure of the algorithm

```
Main Program:
Initialize Population;
while MaxGeneration is not reached do
    while GAperGeneration is not reached do
        select two different parent from population pool;
        create an offspring by using the parents;
        offspring_opt ← LocalSearch(offspring);
        insertToPopulation(offspringopt);
    end while
    while LSperGeneration is not reached do
        select a random chromosome from population pool;
        chromosome_opt ← LocalSearch(chromosome);
        insertToPopulation(offspringopt);
    end while
    while MutationperGeneration is not reached do
        select a random chromosome from population pool;
        pick a random MutationOp ∈ M = {mutate, mutateMerge, mutateDivide};
        chromosome_mut ← MutationOp(chromosome);
        chromosome_opt ← LocalSearch(chromosomemut);
        InsertToPopulation(chromosomeopt);
    end while
end while
```

Population initialization is a crucial component of a GA. Main algorithm is initiated by randomly generating initial population. Then, the reproduction operators, crossover and mutation, are applied on the initial population. LS operations are also applied on the population to further improve the chromosomes in the population. GA and LS operations are individually utilized on the population over and over until the *MaxGeneration* is reached. *MaxGeneration* represents the number of generations that a program runs. The main loop in Algorithm 3.1 presents the steps that are carried out in each generation. Also the activity diagram of this framework can be seen in Figure 3.2.

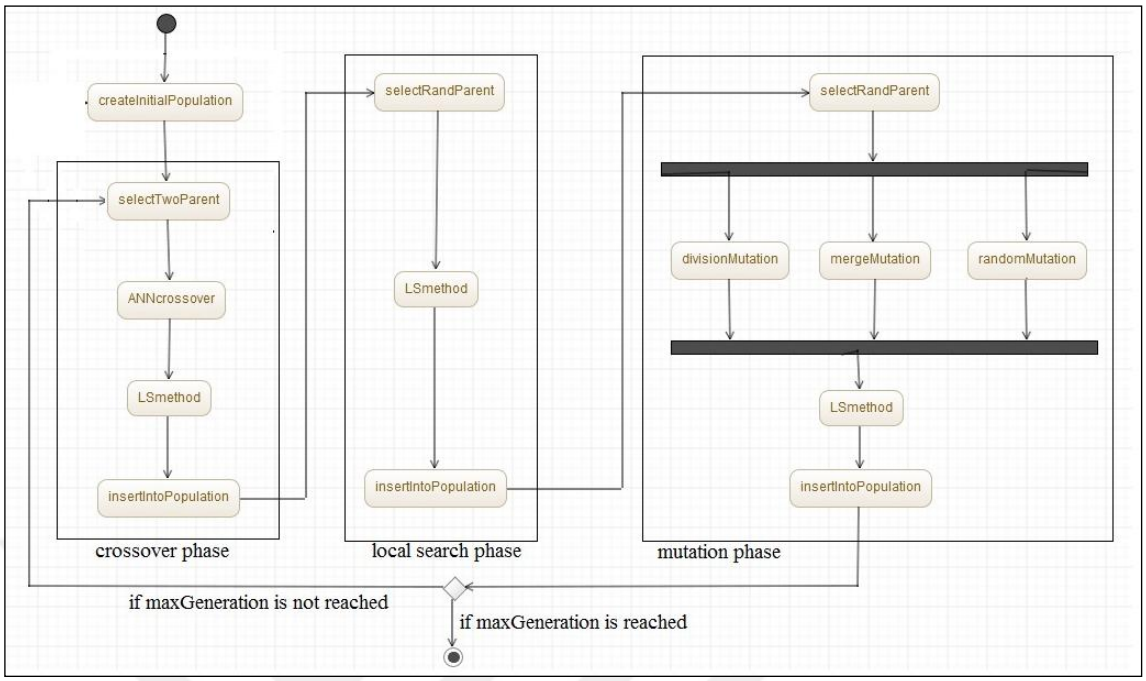


Figure 3.2. Activity diagram of the general framework

The number of times the genetic operators are applied and local search iteration count are determined by the user as input parameters. *GAperGeneration*, *LSperGeneration* and *MutationperGeneration* are parameters denoting the number of genetic and local search operations carried out on the population at each generation. *GAperGeneration* parameter denotes the number crossover operations that take place at each generation. *LSperGeneration* is the number of local search operators and lastly, *MutationperGeneration* parameter represents the number of mutation operations to be carried out again at each generation. Larger values are needed to be used for these parameters for some benchmark problems that are harder to solve. The benchmark problems that are denser than the others and that contain more vertices form the hard instances in the set. Such instances require more computation time, since more genetic and local search operators are applied on them. However, increasing the number of mutation operations utilized may affect the search process in a destructive way. On the other hand, keeping the number of mutations low may result a local minima. Therefore, several experiments need to be carried out in order to fine tune the mutation amount needed during the genetic search.

In Genetic operators are used to provide high variety chromosomes in the population. The chromosomes that are produced by the genetic operators are improved by using local search. Local search improves the convergence in the search process, since it covers the lack of fine tuning in GAs. Basic structure of the main algorithm utilized in this thesis can be seen in Algorithm 3.1.

Algorithm 3.1, it can be seen that the crossover operator is applied on the population at the beginning of each generation. First, two candidate chromosomes are selected from the current population. Tournament selection is used as the selection method. Details of tournament selection are explained in Section 3.3. The candidate chromosomes that are selected for crossover operation are called parent chromosomes.

After the selection process, the parent chromosomes are sent to the crossover function to produce the new chromosomes. Several crossover methods has been implemented and tested in this study in order to determine the most suitable method for the problem at hand.

The new chromosome that is produced by the crossover operation is sent to local search function. The fine tuning of the produced chromosome is achieved by the local search process. Hence, the chromosome is further improved by the local search operator. Then, the chromosome is sent to *insertToPopulation* function. The function determines whether the chromosome will be a member of the population or not. The chromosomes are checked by this method and the ones that met the criteria to be in the population are added to the population, while the others are discarded.

The elite portion of the population is called pareto front. The Pareto front is composed of chromosomes that are not dominated by any other chromosome in the population. Pareto front is formed and managed by the *insertToPopulation* function. The chromosomes that are eligible to enter the population are tested also in terms of the pareto front list. Members of the pareto front list are treated specially and they are guaranteed to exist in the population until a chromosome that dominates them is created by the search operators.

After the crossover phase is finished, local search phase is started on the population. In this phase, a random chromosome is selected from the population and the chromosome is

improved in terms of both bandwidth size and conflict number at the same time. When the predefined iteration count is reached in the local search method, the improved chromosome is sent to the *insertToPopulation* function again. The chromosome might be placed in the pareto front or into the standard population, or it may be completely discarded based on the fitness values.

Lastly, mutation operators are applied on the population before a single generation is completed. Again, a random chromosome is picked from the population and it is sent to the mutation function. The mutation operator that is going to be applied is chosen randomly among a set of different mutation operators. Then, the chosen operator is applied on the chromosome. The mutation operator can destroy some of the successfully formed color groups. Therefore, local search method is also utilized on the mutated chromosomes to avoid the negative effects of this operation. Again, *insertToPopulation* function determines whether the mutated chromosome will be added to the population or not.

The procedures that are explained above take place in a single generation. The same cycle is repeated until the maximum generation count is reached. The methods that are briefly described in this section are taken into consideration one by one and they are explained in detail in the following sections.

## 3.2. POPULATION CONTROL

The proposed framework starts with the creation of the initial population as mentioned in the previous section. Various tests have been run to determine how the initial population should be created. Two different creation methods has been utilized and tested in this study. The first one is a greedy method and the other one is a random approach.

In the greedy method utilized to create the chromosomes, the procedure starts with selecting a random vertex in given graph. Then, the available colors that do not cause a conflict with the adjacent vertices are determined and the one with the smallest index is assigned to the vertex. The selected color must be in range of the currently used bandwidth. This procedure is repeated for all vertices until the graph is fully colored. This approach assigns colors as close as possible to the neighbor vertices in the graph. However,

it has been observed that using this greedy approach decreases the diversity of the population and the local search method cannot be effective during the search process. Therefore, it has been decided to use a random approach while creating the initial population of chromosomes. In this approach, randomly chosen vertices are colored with randomly chosen colors in a predefined bandwidth size. The color bandwidth size is an input parameter given by the user. This bandwidth is expected to be larger than the optimal bandwidth size and hence a diversity of coloring schemes is obtained in the initial population.

After the initial population is created, the genetic and local search operators are utilized to reproduce new chromosomes. Steady state approach has been utilized in the proposed algorithm. The genetic and the local search operators are applied on a portion of the population. When a new chromosome is reproduced, it is added to the population if it meets some predefined criteria.

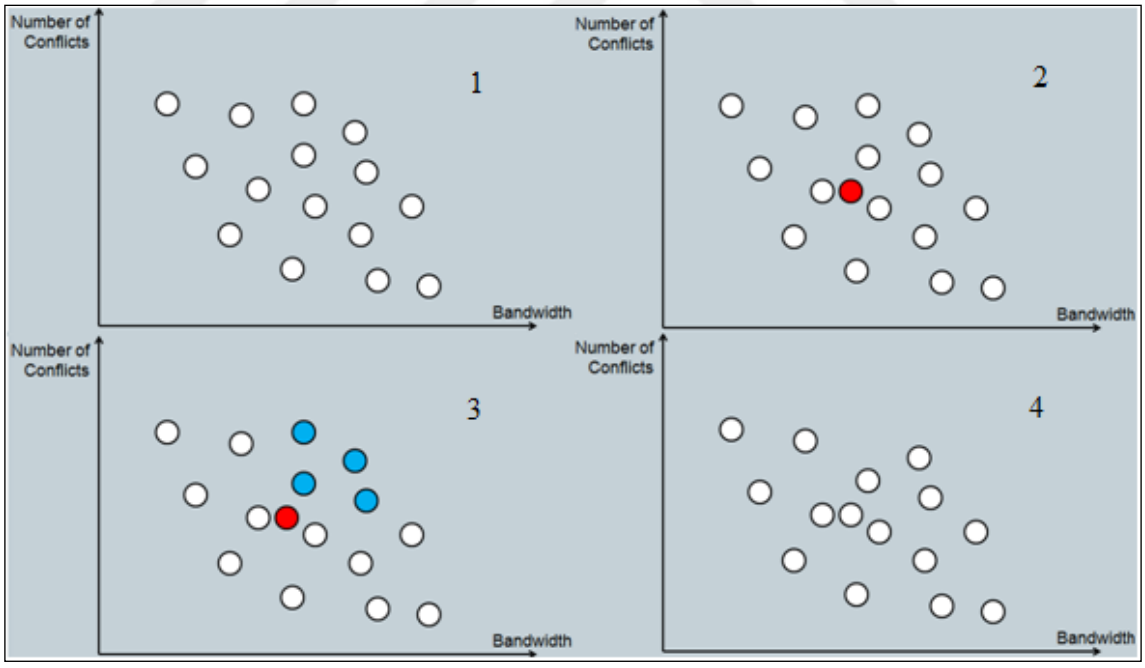


Figure 3.3. An example of adding a chromosome into the population

An example of adding a chromosome into population can be seen in Figure 3.3. As mentioned before, LS and GA work separately and creates new candidate solutions that can be added to the population. These candidates are tested to see if they are adequate or

not. When a candidate is added to the population, it replaces an existing chromosome. In Figure 3.3, the candidate chromosome is denoted with the red circle. When a candidate can dominate some elements in the current population, it has the right to enter the population. In the figure, the blue circles denote the chromosomes that are dominated by the new element. In this case, one of these dominated elements is removed from the population randomly. If the candidate cannot dominate any element, there is still a chance for it to replace a randomly chosen element in the population.

It is important to use some restrictions in order to determine which chromosomes will be added to the population. The population control method utilized is explained in Figure 3.4. A region in the search space is defined by using the current population. Area of the region is calculated by using two input parameters. These are *bandwidthMaxError* and *conflictMaxError*. The *bandwidthMaxError* determines the minimum bandwidth that is allowed in the population. The minimum bandwidth is calculated by subtracting the *bandwidthMaxError* from the current best bandwidth. The *conflictMaxError* determines the maximum conflict number that is allowed in the population. The maximum conflict number is calculated by adding the *conflictMaxError* to the conflict number of the best chromosome in the population. The coordinates of the region depend on the best chromosome in the current population. The best chromosome is the one that has the minimum conflict number. The bottom right hand side corner of the area corresponds to the best chromosome in the population.

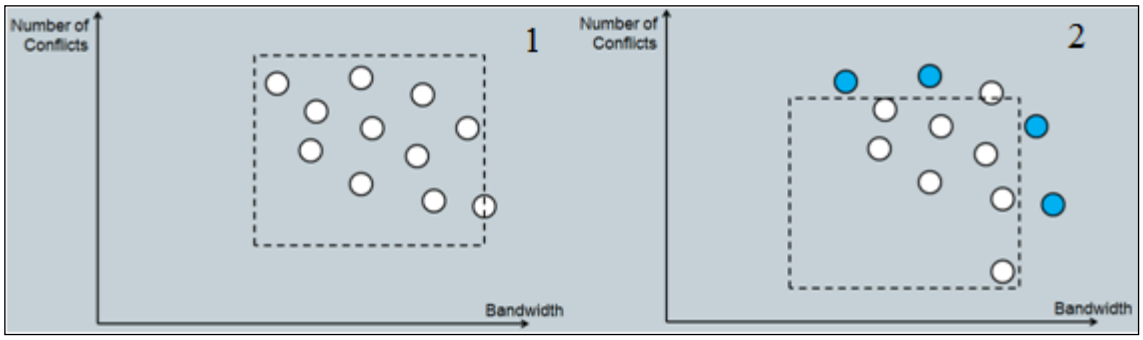


Figure 3.4. The acceptance region

In Figure 3.4, white circles represent the chromosomes that are in the current population. The dotted lines show the boundaries of the region defined for the population. If a newly

created chromosome is in this region in terms of bandwidth size and conflict number, then it is added to the current population. If a chromosome with a lower bandwidth size and lower conflict number compared to the best chromosome is found, then the restricted region is reformed corresponding to this new best chromosome. This situation is demonstrated in Figure 3.4-2. When the region is reformed, there might be some chromosomes that get out of the newly defined region. The chromosomes that are not eligible to enter the population are represented with blue circles in the figure. These chromosomes are not immediately discarded from the population. However, newly created chromosomes replace them in latter generations.

The method provides the use of a concentrated search space. Discarding the chromosomes with too low bandwidth values or too high conflict number increases the speed of the search process.

### 3.2.1. Pareto Effect

Multi objective nature of the BMCP requires special treatment for pareto front which is the elite part of the population. The chromosomes that cannot be dominated by other chromosomes have higher chance to produce better offspring. Different approaches are used in multi objective applications to protect such chromosomes. In this thesis, an approach inspired from the methods in [21, 24], is used for this purpose.

Superior chromosomes that cannot be dominated by other chromosomes should be retained in the population throughout the generations. The building blocks that make these chromosomes superior are expected to help the population to converge the global optimum. Hence, the superior chromosomes should be preserved in the population. This task is handled by a pareto efficiency algorithm. As mentioned above, the method utilized is similar to the approaches in [21, 24]. In [21], more than one pareto front list are utilized during the GA run. In our framework, one global pareto front list is used to handle the elitism in the population like in [24].

Pareto efficiency algorithm is used to create the pareto front list which consists of the non-dominated chromosomes of the population. Pareto front elements are considered as elites.

The pareto front elements should be secured while inserting new chromosomes to the population.

When a new chromosome deserves to be inserted to the population, first it is tested to determine if the chromosome belongs to pareto front or not. In this procedure, the newly created chromosome is compared to all members of the pareto front. If the members of the pareto front cannot dominate the new chromosome in terms of bandwidth and conflict number, the new chromosome is added to the pareto front. After this step, chromosomes that are the dominated by the new chromosome are determined in the pareto front list. Such dominated chromosomes are removed from the list.

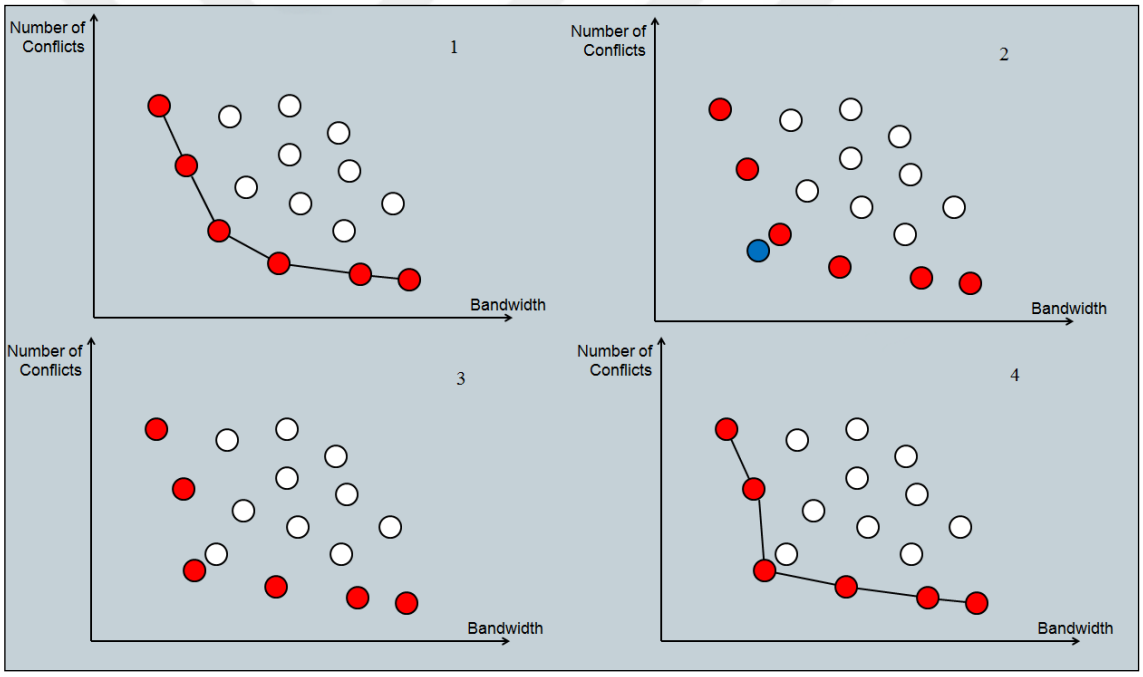


Figure 3.5. An example population and its pareto front

An example population and its pareto front can be seen in Figure 3.5. Red circles represent chromosomes in the pareto list. As seen in the first figure, when the chromosomes in the pareto front are connected together, they form a line in front of the population. The rest of the population is represented by the white circles. There is a new chromosome that is recently added to the population in the second figure. It is represented by a blue circle. This chromosome cannot be dominated by the members of the pareto front list. Thus it is

eligible to enter to the pareto front. It can be seen that there is a chromosome in the pareto front that is dominated by the new chromosome in the third figure. Since this chromosome is dominated, it can no longer be a member of the pareto front list. Hence it should be removed from the list.

Pareto front list consists of the best individuals in the population. These best individuals are better than the rest of the population in terms of bandwidth size and conflict number. Preserving these elements in the pareto front helps convergence of the search process and it becomes possible to find better coloring schemes during the genetic search.

When a newly created chromosome fails to enter the pareto front, it still has the chance to be a member of the standard population. If this chromosome dominates an element in population, it is added to the population by replacing the dominated element. If the newly created chromosome cannot dominate any other elements in the population, it might be still added to the population with a small probability by replacing a randomly chosen chromosome. The probability of entering the population differs according to the operator used to create the new chromosome. The probability value is kept high for the chromosomes that are reproduced by the mutation operators. Mutation operators are rather destructive and using a high probability for this operation adds an extra level of diversity to the population. The probability used for the other genetic and local search operators is low compared to the mutation operator.

Size of the pareto front list should be limited. The size of pareto front is a parameter and it is set to one-third of the population size. If the size of the list is not limited, it rapidly grows and covers the whole population during the search process. It is aimed to have a balanced dispersion among the chromosomes with different bandwidth values. The method used allows us to keep the size of the pareto front limited, while having some degree of diversity in the pareto front list in terms of bandwidth size. The chromosomes in the pareto front are grouped according to the bandwidth value they have. When the size of the pareto front exceeds its limit, a random chromosome is chosen from the most populated group and it is removed from the pareto front. Hence, it is guaranteed to have chromosomes that have different bandwidth values in the pareto front. It is also tested to remove the worst chromosome in the most populated group. However, this removal has created a bias and

disturbed the diversity of the pareto front. Therefore, the element to be removed from the pareto front is chosen randomly from the most populated group.

### 3.3. GENETIC ALGORITHM OPERATORS

Standard genetic operations are very destructive on GCP. The coloring scheme in the chromosomes can be easily destroyed during the recombination process. Hence, standard crossover operations (such as the uniform, one-point or two-point crossover) usually produce chromosomes that have more conflict number or larger bandwidth value than their parents. In this thesis, a customized crossover operation that focuses on determining the critical color groups in the parents is proposed. The method transfers these groups to the offspring as a whole. Other crossover operators are also tested on the benchmark problems in this work. While choosing the best crossover operator, the characteristics of graph coloring problem is considered.

Several methods are used for the selection of parent chromosomes in the literature. The tournament selection method has come to the fore for multi objective problems such as BMCP. The tournament selection method is also used in this thesis.

When the tournament selection method applied to single-objective problem, the first two individuals are selected randomly from the population, and then the chromosome with a higher fitness value is selected as the first parent. One more tournament is carried out to determine the second parent. However, in a multi objective problem, one of the selected individuals may fail to dominate the other in terms of all objectives. In such a case the individuals are considered to be equivalent.

As mentioned before two distinct objective functions are used in the algorithmic framework. The first one is the total bandwidth of the colors used and the second one is the number of conflicts in the current coloring scheme. These two objectives are utilized in the multi-objective tournament selection. In Figure 3.6, let’s assume that the blue and red chromosomes are selected as candidates. In this case we can say that blue chromosome is better since it is better on all objectives and dominates the red one.

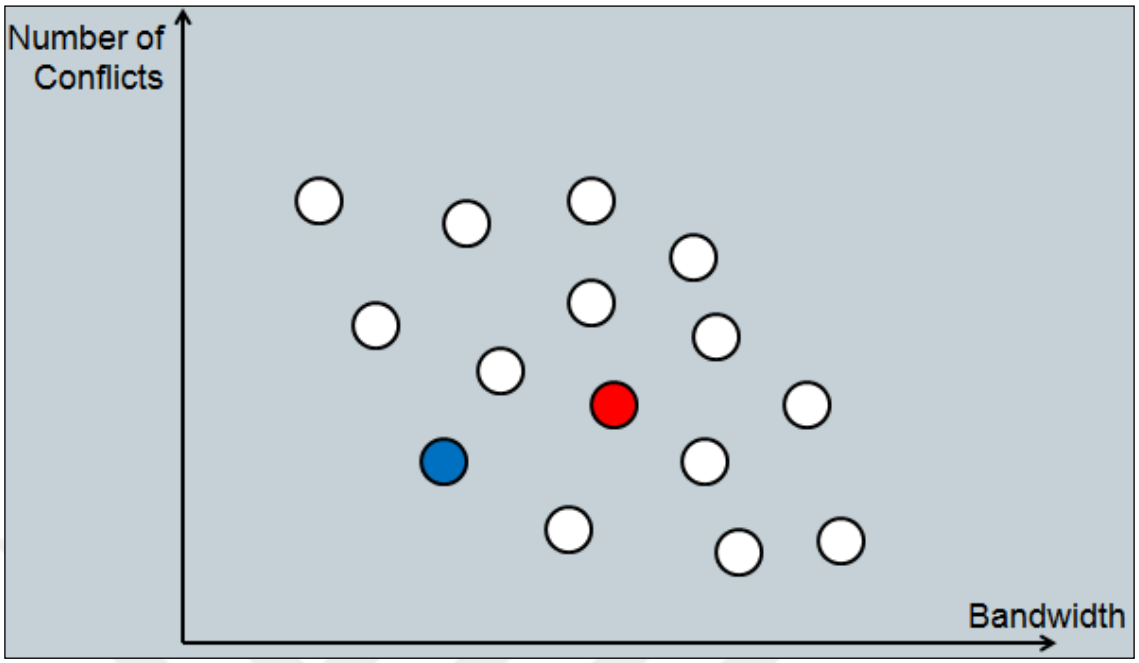


Figure 3.6. Comparison of two chromosomes in the population

In the selection process, two random candidates are selected as in the standard tournament selection. If one of the candidates has better fitness values in terms of both objectives, then it is directly selected like the single objective case. However, if the candidates cannot dominate each other in terms of both objectives, a different criterion is used for the selection. A set of randomly chosen chromosomes is utilized to make a comparison in this case. Each candidate is compared with the elements of this randomly chosen set. The candidate that dominates more elements in the comparison set is selected as a parent for the crossover operation. If the tie is not broken by the comparison set, then the parent is determined randomly among the candidates. This tournament selection is inspired by selection process presented in [34].

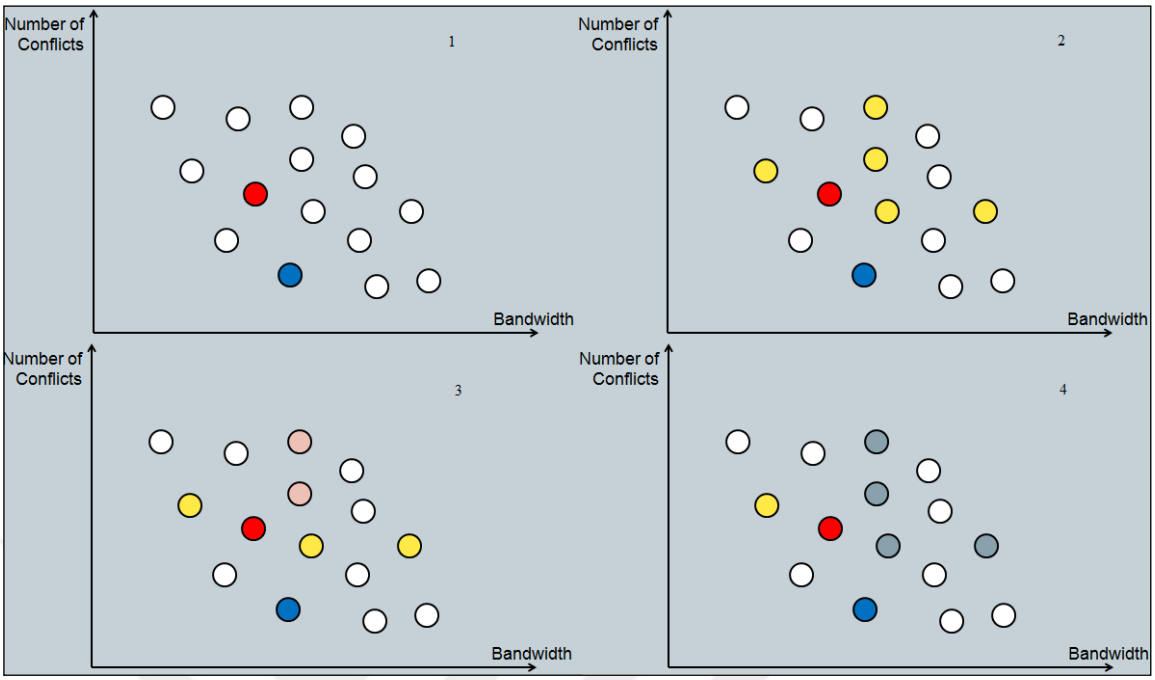


Figure 3.7. Multi objective tournament selection example

In Figure 3.7, a multi objective tournament selection example can be seen. The two elements that are chosen for the tournament are denoted by the red and blue circles. We can't say which candidate is better because blue has lower conflict number while red has a better bandwidth value. Let's assume that five nodes are selected as the comparison set. Each candidate is compared to this set and scored depending on how many of them are dominated. For red chromosome only the two chromosomes denoted by the brown color are dominated. The other elements in the comparison set are denoted with yellow color. For blue chromosome the number of dominated elements is four and they are denoted with the grey color. Since blue dominates more, it is selected as the parent.

Another crucial point in GAs is the representation of chromosomes. Group Number Encoding has been used as the representation scheme for the chromosomes in this study. This approach is similar to chromosome encoding used in [20]. In the conventional group number encoding, each vertex is represented by a gene in the chromosome structure. Colors in the coloring scheme are denoted by the numerical values in the genes. However, in BMCP a vertex can have more than one color based on the weight assigned to the vertex. Therefore, a group of genes are reserved for each vertex. The group size is

determined by the weight of the vertex. An example chromosome structure can be seen in Figure 3.8. Bold lines denote the boundaries of the gene groups used for a single vertex. The vertex weights of the graph can be seen in Figure 3.9. The graph structure can be easily represented with the group number encoding method. The first vertex $v_1$ has weight two in the graph. That is why two genes are reserved for this vertex in the chromosome. The values of these two genes five and eight are the indexes of the colors assigned to this vertex. The other vertices are colored according to the values of other genes in the chromosome.

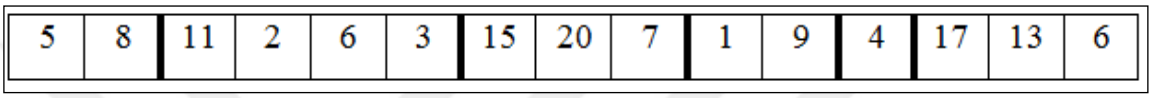


Figure 3.8. An example of chromosome representation

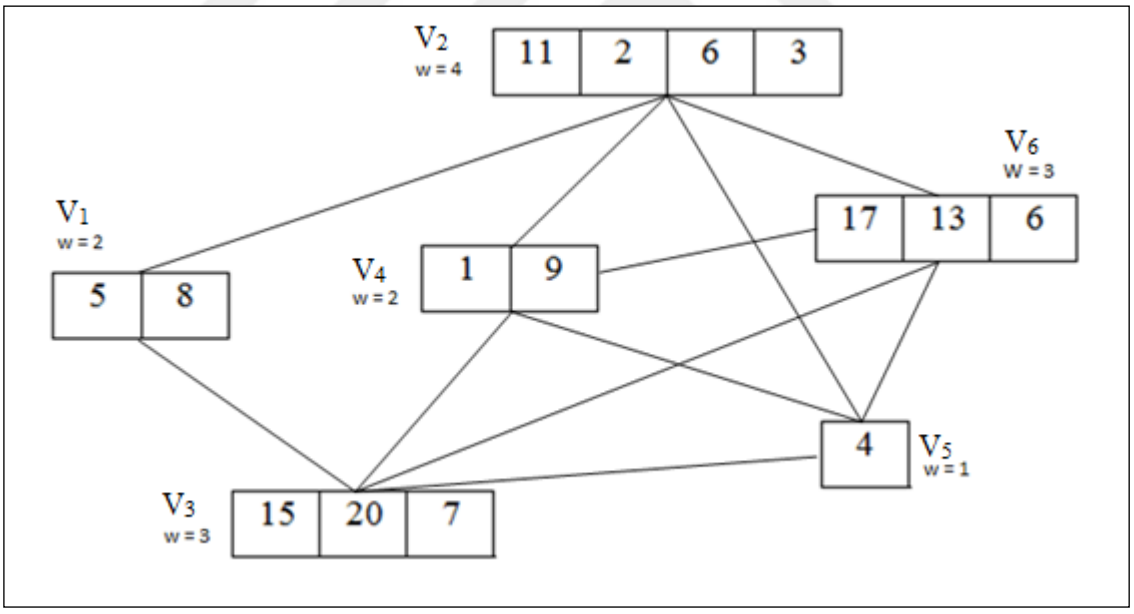


Figure 3.9. Graph representation of the chromosome

### 3.3.1. Crossover Operators

Four different crossover operators are tested in this study. Malaguti and Toth have proposed a novel approach for crossover process in [16]. Two different crossover operators

are developed in this study based on the operator used in [16]. The new crossover operators are called as distance based crossover and vertex based crossover.

In distance based crossover, colors are transferred to the offspring pair by pair. The algorithm of the distance based crossover can be seen in Algorithm 3.2. The first process is the identification of the gene pairs that do not have conflicting colors with each other. Then these colors can be transferred from the parents together. Initially, the first gene is selected, and then the first gene that does not have a conflicting color with this gene is determined. $B_s^1$ and $B_s^2$ represent the set of the color pairs that can be transferred together from the first and the second parent respectively. Then the same process is repeated for the following unselected genes in the chromosome. Thus, all pairs of genes which do not have conflicting colors are determined. Then all the color pairs that can be taken from the first parent are transferred to the offspring. After this step, the same process is repeated for the second parent. While transferring the color pairs from the second parent, the colors already transferred from the first parent should be taken into consideration. The selected color pairs will be transferred to the offspring, if the vertices are not already colored with the colors from the first parent. If the one of the genes that is supposed to be colored with current color pair is already colored, then only one color from the pair is transferred. On the other side, if a conflict arises between the colors transferred from the first and the second parent, the color from the second parent is changed to the next or previous color in the bandwidth. This allows us to pick a color from bandwidth that does not cause any conflict with the previously colored location. When this process ends, remaining uncolored locations are colored with the first available color from the bandwidth, starting from the first color. If there are still uncolored genes after these steps, they are colored with randomly chosen colors from the current bandwidth.

Algorithm 3.2. Distance Based Crossover

```
B^1 represents first parent,
B^2 represents second parent,
B^0 represents offspring;
for i = 0, i < chromosomeSize, i = i + 1 do
    for j = 0, j < neighbor count of i, j = j + 1 do
        if !conflict(B^1 → colorData[i], B^1 → colorData[j]) then
            B_s^1 ← B1 → colorData[i];
            B_s^1 ← B1 → colorData[j];
        end
        if !conflict(B^2 → colorData[i], B^2 → colorData[j]) then
            B_s^2 ← B2 → colorData[i];
            B_s^2 ← B2 → colorData[j];
        end
    end
end
for i = 0, i < chromosomeSize, i = i + 1 do
    for j = 0, j < neighbor count of i, j = j + 1 do
        if B^0 → colorData[i] and B^0 → colorData[j] is not colored then
            B^0 → colorData[i] ← B_s^1[i];
            B^0 → colorData[i] ← B_s^1[j];
        end
    end
end
for i = 0, i < chromosomeSize, i = i + 1 do
    for j = 0, j < neighbor count of i, j = j + 1 do
        if B^0 → colorData[i] and B^0 → colorData[j] is not colored then
            B^0 → colorData[i] ← B_s^2[i];
            B^0 → colorData[i] ← B_s^2[j];
        end
        else if B^0 → colorData[i] is colored then
            if !conflict(B^0 → colorData[i], B^2 → colorData[j]) then
                B_s^1 ← B^1→colorData[j];
            end
        else if B^0 → colorData[j] is colored then
            if !conflict(B^0 → colorData[j], B^2 → colorData[i]) then
                B_s^1 ← B1 → colorData[i];
            end
        end
    end
end
for i = 0, i < chromosomeSize, i = i + 1 do
    if B^0 → colorData[i] is not colored then
        B^0 → colorData[i] ← pick a random color in current bandwidth
    end
end
```

In the testing stage of the distance based crossover, it has been observed that the color pairs are mainly transferred from the first parent. The color pairs from the second parent usually fail to color the uncolored genes due to the conflicts that appear. Then these locations are colored randomly. In this case, the crossover operator behaves more like a mutation

operator. Thus, it was not possible to obtain satisfactory results with this initial crossover operation. Therefore, a new crossover operator is developed to enhance the performance.

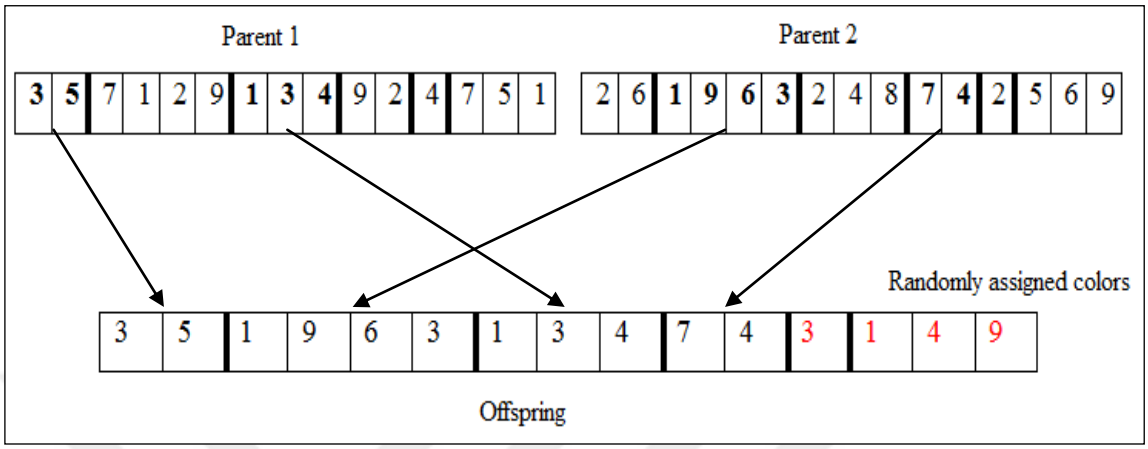


Figure 3.10. Explanation of vertex based crossover

This new crossover operator is called as vertex based crossover. In the vertex based crossover operator, the vertex pairs that do not have conflicting colors are determined. Note that a vertex can have multiple colors in BMCP. Hence, all colors in the corresponding genes are checked against each other in order to determine a vertex pair that does not have any conflicting colors. Then all colors in these vertex pairs are transferred to the offspring instead of the color pairs. The subsets of the vertex pairs are represented as $B_n^1$ and $B_n^2$ in Algorithm 3.3. First of all, the vertex pairs that are colored without having conflicts between them are determined in both parents. Then these vertex pairs are transferred to the offspring in random order. The pairs are selected from different parents in each step. When the process ends, remaining uncolored genes are colored with the first available color from the bandwidth. Finally, if there are still uncolored genes, they are colored again with randomly colors within the bandwidth. The pairs that are transferred in the crossover are chosen evenly from both parents and better results are obtained compared to the distance based crossover. Performance comparison of the two crossover operators can be found in Section 4.2.

An example for the vertex based crossover can be seen in Figure 3.10. The vertex pairs that do not have conflicting colors in each parent are represented as bold numbers. These vertex

pairs are transferred to the offspring. Then, the remaining uncolored vertices are randomly colored. Randomly assigned colors are shown with red numbers.

Algorithm 3.3. Vertex Based Crossover

```
B^1 represents first parent,
B^2 represents second parent,
B^0 represents offspring;
for i = 0, i < nodeSize, i = i + 1 do
    for j = 0, j < neighbor count of i, j = j + 1 do
        if !conflict(B^1→node[i], B^1→node[j]) then
            B^1_pairs ← B1 → node[i];
            B^1_pairs ← B1 → node[j];
        end
    end
end
for i = 0, i < nodeNumber, i = i + 1 do
    for j = 0, j < neighbor count of i, j = j + 1 do
        if !conflict(B1 → node[i], B1 → node[j]) then
            B^2_pairs ← B^2→node[i];
            B^2_pairs ← B^2→node[j];
        end
    end
end
for i = 0, i < nodeSize, i = i + 1 do
    for j = 0, j < neighbor count of i, j = j + 1 do
        k = i % 2 + 1;
        B^0 ← Bk → node[i];
        B^0 ← Bk → node[j];
    end
end
for i = 0, i < chromosomeSize, i = i + 1 do
    if B^0 → colorData[i] is not colored then
        B^0 → colorData[i] ← pick a random color in current bandwidth
    end
end
```

The next crossover operator utilized in this study is called *maxGroupCrossover*. It is based on the method proposed in [7]. This crossover operator aims to transfer the successful color groups to the offspring. Certainly, the successful color groups are the ones which do not cause any conflict in the current coloring scheme. The algorithm used in this operation is presented in Algorithm 3.4. In this crossover, first, the biggest color groups are determined from the two parents selected. These parents are expressed as $B^1$ and $B^2$ in the algorithm. The biggest color group is represented by $B_j^i$ where $i$ is the parent index and $j$ is the group number. Then, the biggest color group from the first parent is transferred to the offspring. The group can be transferred if the genes that belong to this color group are not

already colored by previous operations. Therefore, the colors are removed from both parents whenever they are transferred. This process is repeated until all color groups are transferred to the offspring. There steps of this process is given in Figure 3.11. Here, the largest groups are represented with red, orange and cyan colors. If there are still uncolored genes left after the transfer operation, they are colored with random colors within the current bandwidth.

Algorithm 3.4. Maximum Group Crossover

```
B^1 represents first parent,
B^2 represents second parent,
B^0 represents offspring;
i = 1;
while B^p ≠ ∅ do
    p = i % 2 + 1;
    B_j^p is such a subset that |B_j^p| = max_{1≤m≤k}(|B_m^p|);
    B_j^o = B_j^p;
    B^1 = B^1 \ B_j^p;
    B^2 = B^2 \ B_j^p;
    i = i + 1;
end
for i = 0, i < chromosomeSize, i = i + 1 do
    if B^0 -> colorData[i] is not colored then
        pick a random x between 0 and 1;
        if x == 1 then
            B^0 -> colorData[i] ← pick a random color
                        from in range of B^1's bandwidth;
        else
            B^0 -> renkŞeması[i] ← pick a random color
                        from in range of B^2's bandwidth;
        end
    end
end
```

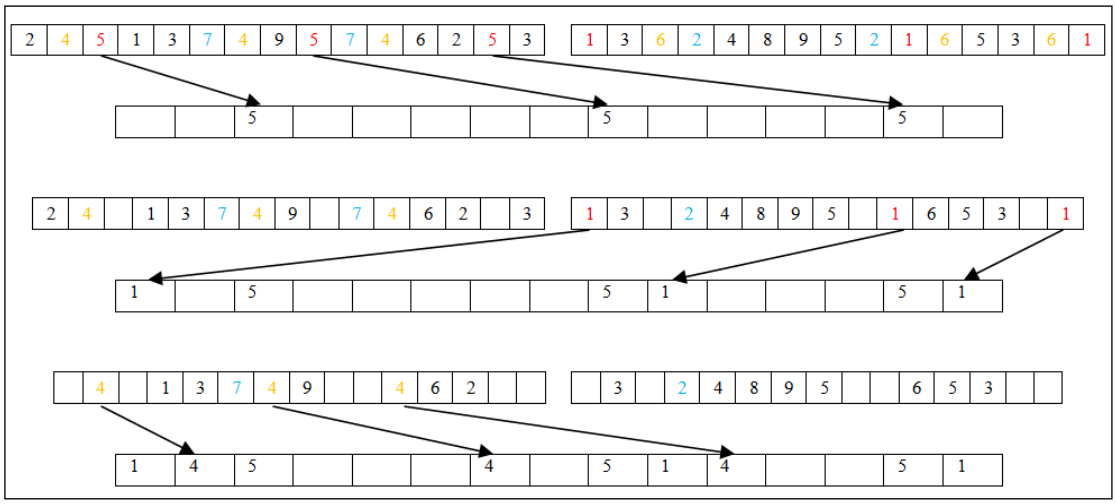

Figure 3.11. There steps of the maximum group crossover

Lastly, new crossover operation that is hybridized with ANNs is also proposed in this thesis. In the genetic search, the crossover operator is carried out in two different ways. At the beginning of the search, the standard uniform crossover is utilized. While the crossover operations are carried out a training data is also created for ANNs. The data holds the parents used in the crossover operation, the gene positions in each parent that are used for coloring the offspring and a label denoting if the operation was successful or not. The crossover operation is labeled as successful if the offspring is better for at least one of the parents in terms of at least one fitness function. When sufficient amount of crossover data is collected an ANN is trained on this data in order to determine the critical color groups that have to be transferred to the offspring as a whole. The standard crossover operation is utilized in the second part of the genetic search. However, three different coloring patterns are created and the trained ANN is utilized to select the pattern that has the highest potential to create a successful offspring. Algorithm of the ANN-Crossover can be seen in Algorithm 3.5.

While creating the coloring patterns, color values are randomly chosen from the parents. The crossover operator reproduces the offspring using these colors. Hence three alternative transfer scenarios are obtained for coloring the offspring. Each scenario form a randomly produced pattern set denoted as $p_i$ in Equation 3.1. Then, the trained ANN is used to determine the pattern that has the highest potential to create a successful offspring as shown in Equation 3.1.Certainly, the offspring is created using the pattern chosen by the

ANN. The chosen pattern represented by $p_{max}$ in Equation 3.1. The representation of ANN-crossover can be seen in Figure 3.12. The pattern in the figure determines the color values that are used to reproduce the offspring. On the other side, the whole ANN-crossover process is explained with a diagram in Figure 3.14.

$$p_{max} = \max\big(ANN(p_i)\big), i = 1,2,3 \tag{3.1}$$

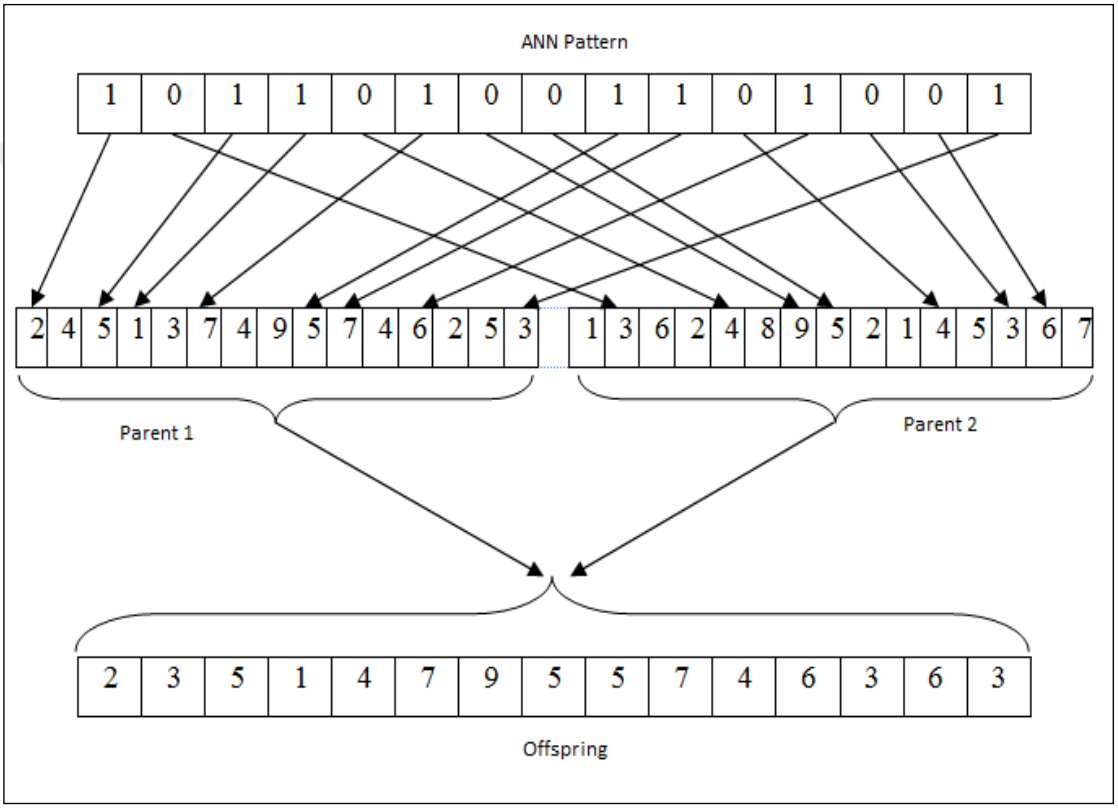


Figure 3.12. Representation of ANN-crossover

Whenever the genetic search starts, after the first 100 generations, the process of collecting training data is also started. This process runs for 50 generations. At the end of these 50 generations, the ANN is trained with the collected data. Then the crossover operation is carried out by using the ANN in the following generations. However, the process of gathering data and retraining the ANN is repeated after each 10000 generations in order to be up to date with new situations that can appear during the search process.

In the training phase, the collected training data is fed to the ANN. The ANN tries to map the coloring patterns to the respective pattern label which denotes if the crossover operation was successful or not. Symmetric sigmoid function is used as activation function in the ANN. The symmetric sigmoid is the sigmoid that is stretched so that the y range is equal to *2* and then it is shifted down by *1* so that it ranges between *-1* and *1*. The sigmoid function is given in Equation 3.2. The ANN produces an output between *-1* and *1* depending on the pattern's potential to create a successful offspring. Results that are closer to one are more likely to produce better offspring. Rprop learning method is used during the training phase in this study. This algorithm reduces the effects of initialization. Also, the learning rate is adaptive in this algorithm.

$$f(x) = \frac{1}{1+ e^{-x}} \tag{3.2}$$

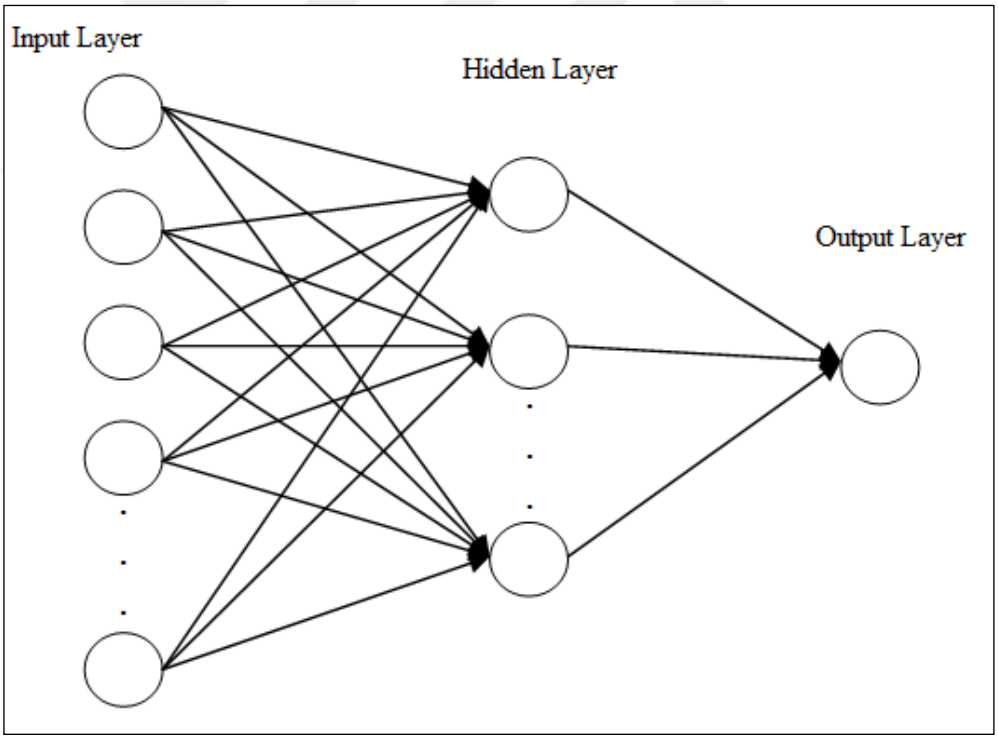


Figure 3.13. ANN architecture

A three layered feedforward neural network is utilized. ANN architecture can be seen in Figure 3.13. Number of neurons used in the input layer is equal to the number of genes used in the chromosome representation. This value changes for each benchmark instance

depending on on the number of vertices in the graph. In Equation 3.3 and 3.4, calculation of the number of neurons used in the input and the hidden layers are given. *I* is the number of neurons in the input layer and *H* in the hidden layer. $n$ represents the number of vertices in the graph. $v_i$ represents a single vertex and $w_{v_i}$ represents the weight value for vertex $v_i$. Half of the number of neurons used in the input layer is used in the hidden layer as seen in Equation 3.4. Lastly, a single neuron is used in the output layer.

$$I = \sum_{i=1}^{n} v_i . w_{v_i} \tag{3.3}$$

$$H = \frac{\sum_{i=1}^{n} v_i . w_{v_i}}{2} \tag{3.4}$$

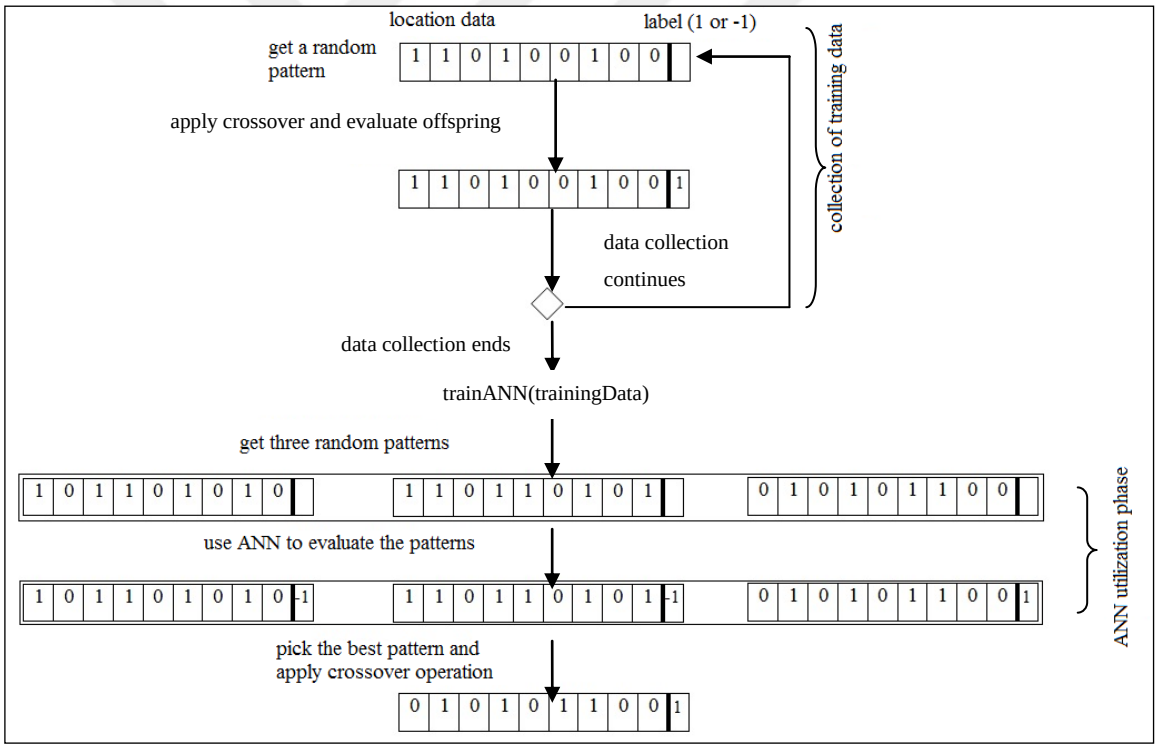


Figure 3.14. Explanation of ANN-crossover

Algorithm 3.5. ANN-Crossover

```
B^1 represents first parent,
B^2 represents second parent,
B^0 represents offspring;
max = 0;
if trainingPhase is active then
    for i = 0, i < chromosomeSize, i = i + 1 do
        randAr[i]→ pick a random value in between 0 and 1;
    end
    for i = 0, i < chromosomeSize, i = i + 1 do
        if randAr[i] == 1 then
            B^0 → colorData[i] ← B^1 [i];
        else
            B^0 → colorData[i] ← B^2 [i];
    end
    if B^0 is better than either B^1 or B^2 do
        annPatternData←randArr;
    end
end
trainAnn(annPatternData);
else
    for i = 0, i < 3, i = i + 1 do
        for i = 0, i < chromosomeSize, i = i + 1 do
            randAr[i] → pick a random value in between 0 and 1;
        end
        output = runAnn(randAr);
        if output > max then
            maxPattern←randAr;
            max = output;
        end
    end
end
for i = 0, i < chromosomeSize, i = i + 1 do
    if maxPattern[i] == 1 then
        B^0 → colorData[i] ← B^1 [i];
    else
        B^0 → colorData[i] ← B^2 [i];
end
```

The group crossover works properly on GCP. Only a single color is assigned to each vertex and all edge weights are set to one in GCP. Hence if you consider two adjacent vertices, it is sufficient to have different colors for a successful coloring on them. Therefore, when the group crossover moves a color group from a parent to the offspring, it is not possible to have some unexpected conflicts with other previously transferred color groups. Therefore, the group crossover performs well on GCP by transferring the color groups which are the building blocks in the parents.

Vertices can be assigned more than one color in BCMP. What is more, the edge weights can be more than one. Hence, certain separations should exist in between the colors assigned to adjacent vertices. Due to these characteristics of the problem, the group crossover does not perform well on BCMP. Whenever, a color group is transferred to the offspring, conflicts may arise due to other color groups already transferred from the second parent. It is not possible to consider color groups as building blocks in this problem. However, there might be some critical regions the graph which is difficult to color due to the edge weights and vertex weights that exist in that region. In BMCP coloring of such regions can be considered as the basic building blocks. Determination of such critical regions is carried out by the ANN crossover in this study. The ANN training used for each benchmark instance can determine the critical graph regions that can easily create conflicts and the approach used enables the crossover operation to transfer all of the colors used in such a critical region from the same parent. Therefore, the ANN crossover outperforms the maximum group crossover on BMCP.

### 3.3.2. Mutation Operators

Three mutation operators are also utilized for the reproduction process. In the first one, a random gene is selected and its color is reassigned to a randomly selected new color within the current bandwidth. Also, two other operators are designed to merge and divide color groups. The algorithm of the merge mutation can be seen in Algorithm 3.6. The merge operator tries to reduce the bandwidth size by merging randomly chosen two color groups into one. The divide operator is in contrast to the merge operator. The algorithm of the division mutation is given in Algorithm 3.7. It tries to increase the bandwidth size by dividing one randomly chosen group into two groups. Two different methods are used in this process. The first method selects the second color group within the current bandwidth range and the second method chooses a color group that is outside the current bandwidth by a specific margin. In many tests, this margin is set as 3 in order not to increase the bandwidth by a big margin. The mutation amount applied in each generation is set as 20 per cent of the population. All the mutation operators have an equal chance to be selected when the mutation process starts.

Algorithm 3.6. Merge Mutation

```
Pick a random k0 color from the current bandwidth;
while(k0 != k1) do Pick a random k1 color from the current bandwidth;
for i = 0, i < chromosomeSize, i = i + 1 do
    if chromosome-> colorData[i] == k1 then
        chromosome-> colorData[i] = k0;
    end
end
```

Algorithm 3.7. Division Mutation

```
pick a random x between 0 and 1;
pick a random k0 color from the current bandwidth;
if x == 0 then
    while(k0 != k1) do Pick a random k1 color from
                                        the current bandwidth;
else
    extraBandwidth ← pick a random x between 1 and 3;
    k1 ← maksimum bandwidth + extraBandwidth;
    chromosomebandwidth ← chromosome → bandwidth + extraBandwidth;
end
for i = 0,i < chromosomeSize,i = i + 1 do
    pick a random y between 0 and 1;
    if chromosome → colorData[i] == k0 and y == 1 then
        chromosome → colorData[i] = k1;
    end
end
```

## 3.4. LOCAL SEARCH OPERATOR

It is a commonly used method to compensate the stochastic framework presented with genetic algorithms by local search operators. In this thesis, a local search operator is used to improve the individuals produced by the genetic algorithm operators. This operator tries to improve the chromosomes by recoloring the conflicting colors with other colors in the current bandwidth. The individuals that will be improved by the local search method are chosen randomly from the population. All chromosomes have the same probability of being selected. A copy of the selected individual is sent to the local search function. The local search function repeatedly changes the colors in the selected chromosome to reveal new coloring schemes. If an improvement can be achieved, then the original chromosome in the population is replaced by the new chromosome.

The recoloring process on the chromosomes is repeated multiple times as mentioned above. The number of steps used for this process is low at the beginning of the search

process. However, the number of iterations in the process increases towards the end of the search process. The number of iterations can also be set to higher values for the graphs which are denser and which has more vertices.

Both the bandwidth size and the number of conflicts in a chromosome are tried to be improved by the local search operator. Local search uses a maximum bandwidth size which is an input parameter of the local search function. Genes that have conflicting colors with other genes or that are outside the input bandwidth size are called as problematic genes. A gene is selected among these problematic genes and it is reassigned to a non-conflicting color within the current bandwidth interval. There are two stopping criteria used in local search operator: The first one occurs when the max iteration count is reached for the operator. The second one occurs when all problematic vertices are eliminated. The resulting chromosome is inserted into the population using the population control algorithm described before. A single iteration of local search method can be seen in Algorithm 3.8.

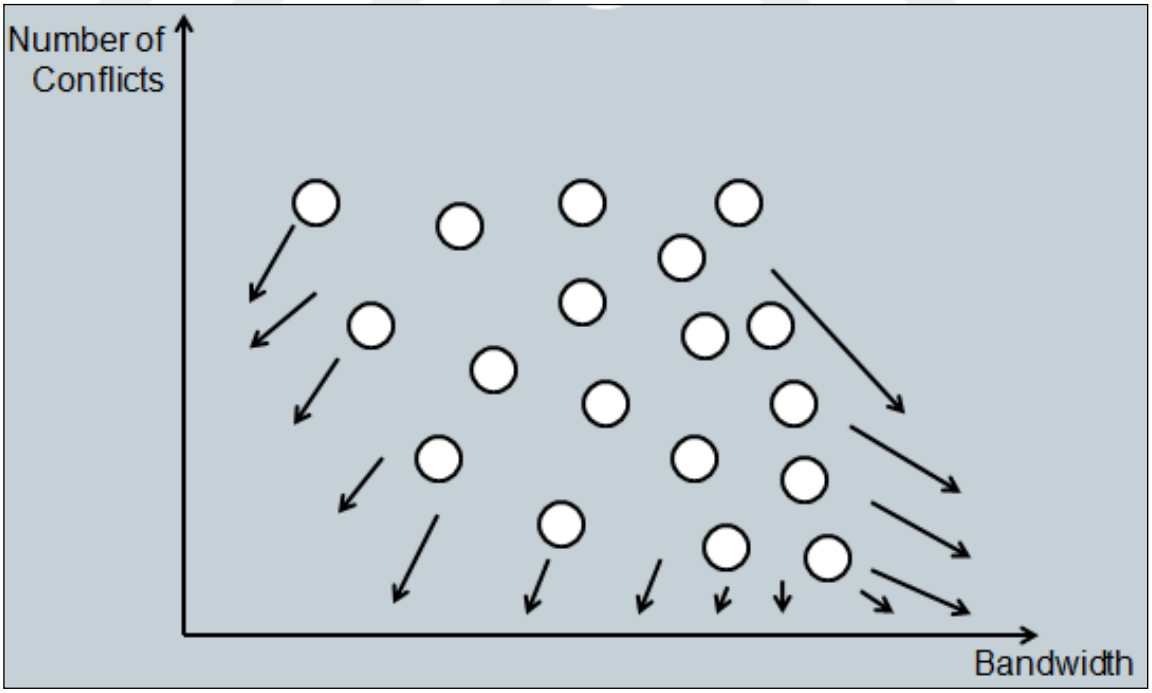


Figure 3.15. Population movement in both directions

Population movement can happen in both directions, the number of conflicts and the bandwidth. This is shown in Figure 3.15. However if a perfect solution with no conflict is found, it is not necessary to search for larger bandwidths anymore. However, if no perfect solution exists in the population, the local search method should be able to increase the color bandwidth in order to reduce the number of conflicting colors.

Algorithm 3.8. Local Search Operator

```
conflictNumbermax ← 0;
colorTocarry ← 0;
newColor ← 0;
diff max ← 0;
for i = 0,i < chromosomeSize,i = i + 1 do
    for j = 0,j < neighbor count of i,j = j + 1 do
        if conflict(chromosome→colorData[i], chromosome→colorData[j])  then
            conflictNumber[i] + +;
        end
    end
    if conflictNumber[i] > conflictNumber_max then
        conflictNumber max ← conflictNumber [i];
        colorTocarry ← i;
    end
end
newConflictCount ← 0;
for k = 0,k < current bandwidth,k = k + 1 do
    chromosome → colorData[colorTocarry] ← k;
    for j = 0,j < neighbor count of chromosome → colorData[colorTocarry],j = j + 1 do
        if conflict(chromosome → colorData[colorTocarry],chromosome → colorData[j])
then
            newConflictCount + +;
        end
    end
    diff ← chromosome → colorData[colorTocarry] – newConflictCount;
    if diff > diff_max then
        diff max ← diff;
        newColor ← k;
    end
end
chromosome → colorData[colorTocarry] ← newColor;
```

As mentioned in the previous paragraph, local search method uses a maximum bandwidth value which is a input given to the function. This input is calculated based on the best chromosome in the population. The best chromosome is the chromosome with the smallest bandwidth that does not contain any conflicts in its coloring scheme. If the best chromosome has no conflicts, then maximum bandwidth for local search operator is set randomly one to four levels lower than the bandwidth of this chromosome. If the best chromosome still has some conflicts, local search operator uses a bandwidth that is again randomly set as one to four levels larger than the current bandwidth.

# 4. EXPERIMENTAL RESULTS

## 4.1. BENCHMARK PROBLEM SET

GEOM benchmarks, generated by Michael Trick [12], are used as BMCP test instances in this study. In the GEOM benchmarks, the vertices of a graph are represented by points that are uniformly distributed in a square by $10000 \; x \; 10000$. Vertices that are closer to each other less than a previously determined threshold have an edge connection in between. The distance between the points determines the separation distances associated to the edges. Number of colors in a vertex is determined by picking a random number from the interval [1,r], where r can get 10 as maximum value and all edge weights are set to 10. The smallest test instances have 20 vertices and new test instances are created by increasing the number of vertices in tens. The largest instance that exists in the set has 120 vertices. Each instance is named as GEOM*n*a and GEOM*n*b where n is the number of vertices that exist in that instance and the instances that have a name that ends with the "b" letter contain denser graphs compared to the instance names that end with the "a" letter.

## 4.2. CONTRIBUTION OF THE OPERATORS

Different genetic and local search operators are designed and used in the algorithmic framework proposed in this thesis. The contribution of each operator is determined throughout some experiments conducted in the study. According to the analysis made, the solutions obtained at the end of the genetic search are mainly constructed by the mutation and local search operators. On the other side, it has been observed that the best solutions can be achieved whenever the crossover operator is also included in the search process. It can be claimed that the crossover operator does not take a important role for the construction of the solution, but the operator still should be used for improving the quality of the solutions obtained.

In the experiments, it was observed that a large portion of the improvements on the chromosomes are provided by the hill climbing operator. Therefore, the hill climbing is applied on all individuals produced by the mutation and crossover operations.

In the experiments, the effect of applying mutation operator on the individuals produced by the crossover operator is also tested. However, it has been observed that performing mutation operator on these offspring results in too much distortion on the individuals and does not contribute to get better results.

The impact of using mutation operator on the individuals produced by the crossover operation can be seen in Table 4.1. As seen in the table, the percentage of producing offspring better than the parents is higher when only crossover is used before the hill climbing process compared to the case where crossover and mutation operators are used together. An individual is better than the other one if it has a smaller bandwidth and smaller number of conflicts in the coloring scheme it represents. If two individuals have the same value for one of the fitness functions, then the individual which has lower value on the second fitness function is considered as the better individual.

Table 4.1. Affect of the mutation after crossover operation

| **Utilized Framework** | **The Percentage Of Producing Offspring Better Than The Parents (%)** |
|---|---|
| Crossover + Local Search | 29.17 |
| Crossover + Mutation + Local Search | 23.82 |

Performance comparison of the distance based crossover and the vertex based crossover on three selected benchmark instances can be seen at Table 4.2. The vertex based crossover obtains better results compared to the distance based crossover. The coverage rate [35] is a commonly used performance evaluation metric for MOGAs. This rate determines how two different algorithms' pareto fronts dominate each other. If the rate is equal to one, this means that all the members of the pareto front created by the first algorithm dominate the members of the pareto front created by the second algorithm. The coverage rate obtained by the vertex based crossover operator is either equal to one or it is very close to one on the benchmark instances used in the experiments.

Table 4.2. Comparison between vertex based crossover and distance based crossover

| **Benchmark Problem Set** | **Coverage Rate(Vertex Based / Distance Based)** | |
|---|---|---|
| | **Average** | **Median** |
| GEOM50 | 0.85 | 0.86 |
| GEOM50a | 0.79 | 0.83 |
| GEOM50b | 1 | 1 |

The performance comparison between vertex based crossover operator and the other operators used in this study can be seen in Table 4.3. The distance crossover operator is not included in this comparison, since the vertex based crossover operator outperforms it.

Table 4.3. Probability of creating better individuals of the crossover operators

| **Crossover Operator** | **The Probability Of Producing Better Offspring In Terms Of Only One Fitness Value (%)** | **The Probability Of Producing Better Offspring In Terms Of Both Fitness Functions (%)** |
|---|---|---|
| Vertex based crossover v1 | 3.27 | 0.79 |
| Vertex based crossover | 0.1 | 0.08 |
| Maximum group crossover | 3.59 | 1.82 |

In the vertex based crossover, the colors are transferred to the new individuals after determining the appropriate pairs of vertices. The approach aims to minimize the conflicts in the offspring. The performance of a different version of vertex based crossover is also presented in the table. The edge weights are also considered and coloring constraints are not violated while transferring the colors in the vertex based crossover operator. The edge weights are not taken into consideration in vertex based crossover-v1. When this method is used, the diversity in the population is increased.

The names of the crossover operators are given in the first column of the Table 4.3. In the second column, the probability of producing better offspring in terms of only one fitness value is given for the operators. Lastly, the probability of producing better offspring in terms of both fitness functions is given in the third column. The probability producing an offspring better than the parents in terms of both fitness functions is low for all operators.. However, this is not an unexpected situation, since the crossover operators are quite destructive for grouping problems. The maximum group crossover has the highest performance compared to the other crossover operators as seen in the table.

Further tests are performed on the ANN-crossover which is designed to increase the performance of the crossover operation. The crossover operator is considered as successful again if it reproduces an offspring that has a better value for one of the fitness functions without resulting a declination of the other fitness function value. Comparison of ANN-crossover and max group crossover are given in Table 4.4. The ANN-crossover outperformed the max group crossover in all the problem instances that exist in the GEOM test suite as seen in Table 4.4.

Table 4.4. Success rate comparison between max group crossover and ANN crossover

| Benchmark Problem Set | Success Rate Of The Crossover Operator (%) | |
|---|---|---|
| | Max Group Crossover | ANN Crossover |
| GEOM20 | 1,62 | 9,45 |
| GEOM20a | 1,05 | 5,42 |
| GEOM20b | 3,16 | 6,47 |
| GEOM30 | 2,21 | 8,03 |
| GEOM30a | 0,93 | 8,61 |
| GEOM30b | 4,97 | 9,86 |
| GEOM40 | 2,00 | 7,35 |
| GEOM40a | 2,13 | 6,90 |
| GEOM40b | 3,56 | 5,61 |
| GEOM50 | 0,99 | 5,29 |
| GEOM50a | 2,28 | 7,67 |
| GEOM50b | 3,20 | 5,67 |
| GEOM60 | 0,97 | 9,24 |
| GEOM60a | 2,03 | 6,22 |
| GEOM60b | 4,73 | 7,84 |
| GEOM70 | 2,60 | 6,17 |
| GEOM70a | 1,39 | 8,13 |
| GEOM70b | 2,74 | 6,11 |
| GEOM80 | 3,41 | 8,29 |
| GEOM80a | 2,05 | 6,09 |
| GEOM80b | 2,63 | 6,02 |
| GEOM90 | 3,44 | 8,64 |
| GEOM90a | 2,41 | 9,49 |
| GEOM90b | 2,42 | 5,42 |
| GEOM100 | 0,91 | 7,45 |
| GEOM100a | 3,12 | 6,70 |
| GEOM100b | 4,05 | 6,64 |
| GEOM110 | 1,76 | 8,61 |
| GEOM110a | 3,07 | 7,44 |
| GEOM110b | 2,35 | 8,62 |
| GEOM 120 | 3,49 | 7,31 |
| GEOM120a | 2,77 | 5,90 |
| GEOM120b | 3,68 | 8,33 |

The effect of the ANN-crossover on the proposed framework is also analyzed in terms of best solutions that can be obtained on the benchmark instances. The best solution generated at the end of the search is the individual that colors the graph successfully with minimum bandwidth value. Certainly, a successful coloring of the instance does not contain any

conflicts. The ANN-crossover achieved improvements on the minimum bandwidth values for the successful colorings, too. The best and average bandwidth values that are obtained for a group of runs on the test instances can be seen in Table 4.6. and Table 4.7. As seen in the table, the crossover operator improved the best bandwidth value in 19 problem instances. Worse results are obtained only in five problem instances by the ANN-crossover compared to the standard group crossover. Lastly, the same results are obtained in the remaining nine problem instances. The average bandwidth values are calculated using the results of 10 different runs on each instance. The ANN-crossover operator improved the average bandwidth value in 30 problem instances. The same result is obtained in only one of the problem sets. Lastly, ANN-crossover has a worse average in only two problem instances. Detailed test results are given in Appendix B section.

Parameters that are used in the tests are given in Table 4.5. The population size is set to 120 in these tests. The tests are run for 30000 generations. 40 LS operations and 30 crossover operations are applied on the population in each generation. Mutation operation count is set to 20 per cent of the population size. In LS operations, maximum iteration count starts from 20 and linearly increases up to 40 throughout the run.

Table 4.5. Parameters used in the tests

| **Population Size** | **Max Generation Count** | **LS Operation Count** | **Crossover Operation Count** | **Mutation Operation Count** | **Min LS Iteration Count** | **Max LS Iteration count** |
|---|---|---|---|---|---|---|
| 120 | 30000 | 40 | 30 | 20 per cent of the population size | 20 | 40 |

Table 4.6. The best bandwidths obtained by the ANN-crossover

| Benchmark Problem Set | Best | |
|---|---|---|
| | Max Group Crossover | ANN crossover |
| | Bandwidth | Bandwidth |
| GEOM20 | 150 | 150 |
| GEOM20a | 170 | 170 |
| GEOM20b | 44 | 44 |
| GEOM30 | 160 | 160 |
| GEOM30a | 213 | 212 |
| GEOM30b | 77 | 77 |
| GEOM40 | 167 | 167 |
| GEOM40a | 216 | 214 |
| GEOM40b | 75 | 75 |
| GEOM50 | 225 | 225 |
| GEOM50a | 326 | 326 |
| GEOM50b | 88 | 87 |
| GEOM60 | 258 | 259 |
| GEOM60a | 372 | 367 |
| GEOM60b | 118 | 119 |
| GEOM70 | 275 | 274 |
| GEOM70a | 480 | 479 |
| GEOM70b | 125 | 123 |
| GEOM80 | 393 | 389 |
| GEOM80a | 382 | 376 |
| GEOM80b | 141 | 143 |
| GEOM90 | 339 | 338 |
| GEOM90a | 388 | 387 |
| GEOM90b | 157 | 155 |
| GEOM100 | 425 | 412 |
| GEOM100a | 464 | 460 |
| GEOM100b | 168 | 172 |
| GEOM110 | 392 | 394 |
| GEOM110a | 511 | 504 |
| GEOM110b | 215 | 213 |
| GEOM 120 | 414 | 411 |
| GEOM120a | 569 | 564 |
| GEOM120b | 208 | 203 |

Table 4.7. The average bandwidths obtained by the ANN-crossover

| Benchmark Problem Set | Average | | | |
|---|---|---|---|---|
| | Max Group Crossover | | ANN crossover | |
| | Bandwidth | Time | Bandwidth | Time |
| GEOM20 | 151,9 | 3201 | 151 | 3561 |
| GEOM20a | 171,4 | 4095 | 170,3 | 4978 |
| GEOM20b | 44 | 646 | 44 | 765 |
| GEOM30 | 161,1 | 4239 | 160,4 | 6000 |
| GEOM30a | 216,5 | 9319 | 214,3 | 11742 |
| GEOM30b | 77,7 | 1418 | 77,5 | 1794 |
| GEOM40 | 169,6 | 6427 | 168,4 | 8461 |
| GEOM40a | 221,8 | 14119 | 218,2 | 15900 |
| GEOM40b | 76,9 | 2336 | 76,4 | 2933 |
| GEOM50 | 227,1 | 9299 | 226,8 | 12196 |
| GEOM50a | 338,2 | 30482 | 334,5 | 35254 |
| GEOM50b | 91,2 | 4340 | 88,4 | 4594 |
| GEOM60 | 263,5 | 11280 | 261,4 | 15743 |
| GEOM60a | 376 | 29236 | 371,3 | 36655 |
| GEOM60b | 122,6 | 5960 | 122 | 7425 |
| GEOM70 | 280,2 | 19987 | 277,2 | 23898 |
| GEOM70a | 485,9 | 35394 | 483,8 | 43586 |
| GEOM70b | 127,7 | 7288 | 127,4 | 8912 |
| GEOM80 | 397,5 | 28219 | 393,2 | 32747 |
| GEOM80a | 386 | 37266 | 382,4 | 47315 |
| GEOM80b | 144,6 | 9055 | 144,7 | 11883 |
| GEOM90 | 343,7 | 34378 | 341,5 | 38021 |
| GEOM90a | 394,8 | 42423 | 394,1 | 49780 |
| GEOM90b | 161 | 12017 | 159 | 15982 |
| GEOM100 | 429,2 | 30663 | 422,4 | 42483 |
| GEOM100a | 471,3 | 73922 | 464,5 | 81071 |
| GEOM100b | 174,8 | 14804 | 176,5 | 18471 |
| GEOM110 | 401,3 | 44176 | 398,5 | 60623 |
| GEOM110a | 519,4 | 76800 | 510,1 | 100616 |
| GEOM110b | 218,4 | 17247 | 215,7 | 21874 |
| GEOM 120 | 421,9 | 48918 | 418,3 | 61746 |
| GEOM120a | 584 | 92671 | 573,3 | 116158 |
| GEOM120b | 210,7 | 18362 | 207,8 | 23177 |

Various experiments are carried out to observe the effect of mutation operators used in the study. The results obtained while all mutation operators are simultaneously active can be seen in Table 4.8. All of the operators in the table are equally likely to have the chance to be operated.

Conflict number that exists in the offspring produced by using the standard mutation operator does not get worse with a rate of 6.5 per cent. This rate is much higher compared to the merge and the division mutation operators. The effect of each operator to the individuals is quite different, as shown in Table 4.8. The bandwidth value is expected to increase and shrink by dividing and merging the color groups. This process increases the diversity of the population. The standard mutation operator also helps to avoid being stuck at local minima by changing color of a randomly picked up vertex in the graph. Different tests are also performed to analyze the effect of using different combinations of the mutation operators. It is observed that the best performance is obtained when all operators are used simultaneously. It is not possible to obtain satisfactory results when only a subset of the mutation operators is utilized.

Table 4.8. Performance comparison of the mutation operators

| **Mutation Operator** | **Total Call Count** | **Case of Increased Conflict is Obtained** | **Case of Same or Decreased Conflict is Obtained** | **Percentage of Not Having a Worse Chromosome (%)** |
|---|---|---|---|---|
| Standard Mutation | 103315 | 96584 | 6731 | 6.51 |
| Merge Mutation | 102733 | 60593 | 42140 | 41.01 |
| Division Mutation | 103287 | 4489 | 98798 | 95.65 |

The first column in Table 4.8 shows the name of the mutation operator used in the tests. The second column shows total number of calls for each mutation operator. The number of calls where the conflict number is increased by the operator can be seen in the third column. With the mutation operators, it mainly is aimed to increase the diversity by changing the bandwidth values of the individuals. When the mutation operator is applied

without increasing the conflict number, it is considered as successful. The number of successful mutation operator calls are shown in the forth column. The rate of these successful calls is given in the last column.

The effect of the local search operator is also observed, as well as crossover and mutation operators. The performance of local search operator on four selected problem instances can be seen in Table 4.9. The individuals which are produced by using local search operator are likely to be better than the original individuals in terms of both conflict number and bandwidth value, as seen in the table. It can be claimed that, the local search operator is especially effective on the conflict number and it can reduce the conflicts in the individuals with a high percentage of success.

Table 4.9. Local search experiment

| **Benchmark Problem Set** | **Total Call Count** | **Cases of Same Conflict is Obtained** | **Case of Decreased Conflict is Obtained** | **Improvement Rate (%)** |
|---|---|---|---|---|
| GEOM50 | 600000 | 420311 | 179497 | 29.91 |
| GEOM50a | 600000 | 471649 | 128167 | 21.36 |
| GEOM50b | 600000 | 382219 | 217309 | 36.21 |
| GEOM60 | 600000 | 443587 | 156088 | 26.01 |

The first column in the Table 4.9 shows the total number of local search calls carried out in a single run. The number of the calls where the conflict number does not change is given in the second column. The number of the calls where the conflict number is decreased by the operator is given in the third column, and improvement rates can be seen in the last column. If the conflict number of an individual is decreased by the local search operator, it is considered as an improvement on the individual. Each row in the table presents the results obtained on a different problem instance.

The hybridization of GAs with local search improves the performance of the methodology. To demonstrate the contribution of local search, coverage rate calculation is also utilized

for the local search operator. It has been seen that all the coverage rates are equal to one in favor of the search accompanied by local search. Also the best bandwidth values obtained by pure MOGA and MOGA accompanied by local search can be seen in Table 4.10 on some selected instances.

Table 4.10. Effect of the local search operator on the best bandwidths

| **Benchmark Problem Set** | **The Best Bandwidths** | |
|---|---|---|
| | **Local Search is Utilized** | **Local Search is not Utilized** |
| GEOM50 | 224 | 274 |
| GEOM50a | 320 | 392 |
| GEOM50b | 88 | 104 |
| GEOM70 | 274 | 365 |
| GEOM70a | 472 | 492 |
| GEOM70b | 121 | 158 |

The average best bandwidth and standard deviation obtained throughout different runs on the some instances can be seen in Figure 4.1. The results are obtained by using 20 different runs for each problem instance. As seen in the figure, again the local search operator has an important contribution for establishing better solutions.

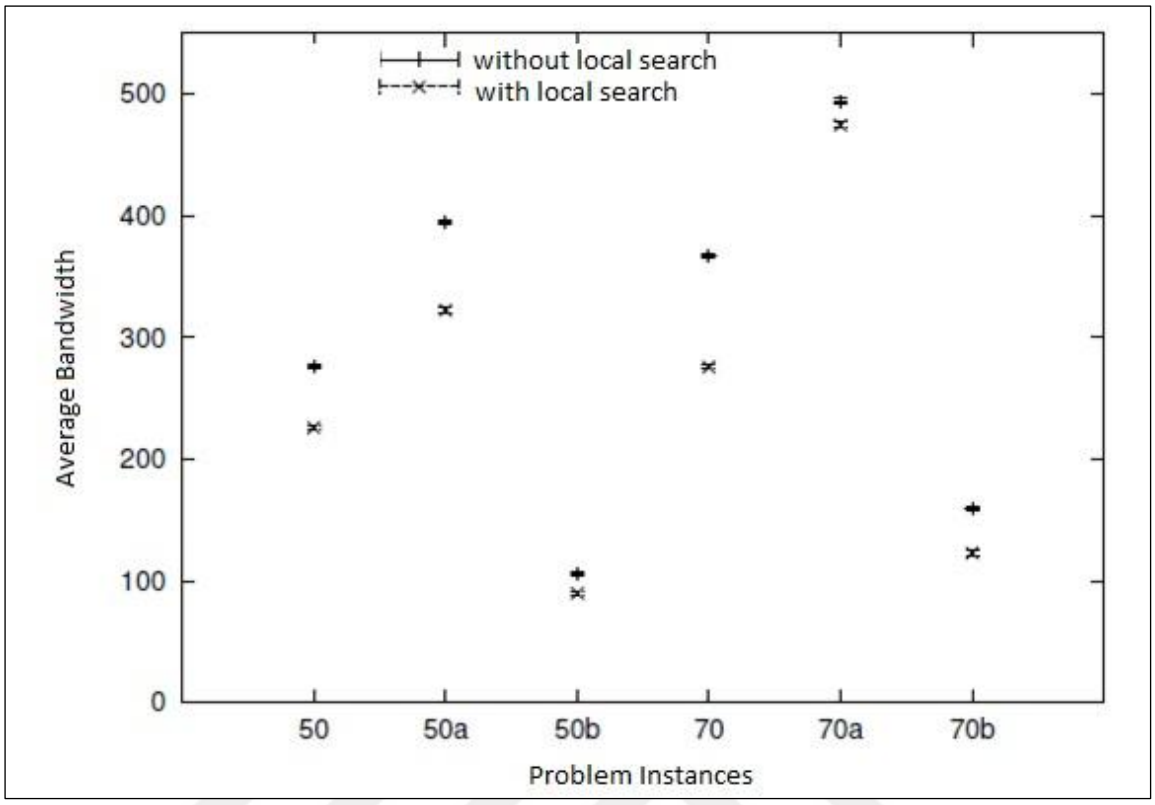


Figure 4.1. Standard deviation of average bandwidth comparison

## 4.3. COMPARISON BETWEEN THE HYBRID MOGA APPROACH AND OTHER ALGORITHMS

Comparison between the hybrid MOGA approach used in this thesis and other three state of the art algorithms can be found in Table 4.11, Table 4.12 and Table 4.13. These three algorithms are proposed in [19] (combination of LS and Constraint Propagation), [15] (combination of TS and SWO) and [16] (combination of GA and TS). All tests are done under Ubuntu machines with 2.66GHz Intel Core Duo CPUs and 2MB RAM. The tests are repeated 10 times and the best results are included in the tables. Names of the benchmarks are given in the first column. The second column shows the results obtained by the MOGA approach. The next columns show the results obtained by the other frameworks and the improvements obtained by the MOGA approach compared to other frameworks. It is difficult to provide a performance comparison between MOGA and other methods in terms of computational time. MOGA performs a parallel search in the pareto front including

solutions with different bandwidths. The other algorithms, utilize only a single bandwidth value. However, in general it can be stated that MOGA is slower compared to [15, 16, 19].

The improvements obtained by the MOGA approach are given in the “improvement” column. The values in these columns are the differences between two frameworks. The positive values in the column show that the MOGA approach has found a better result compared to the other framework. The negative values in the column indicate the cases where the MOGA approach falls behind, when compared to the other frameworks. If the columns hold value of zero, it means that a tie happened between the two approaches.

In these tests population size is set as 120. Max generation count is 30000 and in each generation 40 LS operations and 30 crossover operations are performed. In LS operations, maximum iteration count starts from 20 and linearly increases up to 40 throughout the run.

Table 4.11. MOGA and Prestwich Comparison

| **Benchmark Problem Set** | **MOGA** | **PRESTWICH [19]** | **Improvement** |
|---|---|---|---|
| GEOM20 | 150 | 149 | -1 |
| GEOM20a | 170 | 170 | 0 |
| GEOM20b | 44 | 44 | 0 |
| GEOM30 | 160 | 160 | 0 |
| GEOM30a | 212 | 214 | 2 |
| GEOM30b | 77 | 77 | 0 |
| GEOM40 | 167 | 167 | 0 |
| GEOM40a | 214 | 217 | 3 |
| GEOM40b | 75 | 74 | -1 |
| GEOM50 | 225 | 224 | -1 |
| GEOM50a | 326 | 323 | -3 |
| GEOM50b | 87 | 86 | -1 |
| GEOM60 | 259 | 258 | -1 |
| GEOM60a | 367 | 373 | 6 |
| GEOM60b | 119 | 116 | -3 |
| GEOM70 | 274 | 277 | 3 |
| GEOM70a | 479 | 482 | 3 |
| GEOM70b | 123 | 119 | -4 |
| GEOM80 | 389 | 398 | 9 |
| GEOM80a | 376 | 380 | 4 |
| GEOM80b | 143 | 141 | -2 |
| GEOM90 | 338 | 339 | 1 |
| GEOM90a | 387 | 382 | -5 |
| GEOM90b | 155 | 147 | -8 |
| GEOM100 | 412 | 424 | 12 |
| GEOM100a | 460 | 461 | 1 |
| GEOM100b | 172 | 159 | -13 |
| GEOM110 | 394 | 392 | -2 |
| GEOM110a | 504 | 500 | -4 |
| GEOM110b | 213 | 208 | -5 |
| GEOM120 | 411 | 417 | 6 |
| GEOM120a | 564 | 565 | 1 |
| GEOM120b | 203 | 196 | -7 |

Table 4.12. MOGA and Lim Comparison

| Benchmark Problem Set | MOGA | LIM [15] | Improvement |
|---|---|---|---|
| GEOM20 | 150 | 149 | -1 |
| GEOM20a | 170 | 169 | -1 |
| GEOM20b | 44 | 44 | 0 |
| GEOM30 | 160 | 160 | 0 |
| GEOM30a | 212 | 211 | -1 |
| GEOM30b | 77 | 77 | 0 |
| GEOM40 | 167 | 167 | 0 |
| GEOM40a | 214 | 214 | 0 |
| GEOM40b | 75 | 76 | 1 |
| GEOM50 | 225 | 224 | -1 |
| GEOM50a | 326 | 326 | 0 |
| GEOM50b | 87 | 87 | 0 |
| GEOM60 | 259 | 258 | -1 |
| GEOM60a | 367 | 368 | 1 |
| GEOM60b | 119 | 119 | 0 |
| GEOM70 | 274 | 279 | 5 |
| GEOM70a | 479 | 478 | -1 |
| GEOM70b | 123 | 124 | 1 |
| GEOM80 | 389 | 394 | 5 |
| GEOM80a | 376 | 379 | 3 |
| GEOM80b | 143 | 145 | 2 |
| GEOM90 | 338 | 335 | -3 |
| GEOM90a | 387 | 382 | -5 |
| GEOM90b | 155 | 157 | 2 |
| GEOM100 | 412 | 413 | 1 |
| GEOM100a | 460 | 462 | 2 |
| GEOM100b | 172 | 172 | 0 |
| GEOM110 | 394 | 389 | -5 |
| GEOM110a | 504 | 501 | -3 |
| GEOM110b | 213 | 210 | -3 |
| GEOM120 | 411 | 409 | -2 |
| GEOM120a | 564 | 564 | 0 |
| GEOM120b | 203 | 201 | -2 |

Table 4.13. MOGA and Malaguti Comparison

| **Benchmark Problem Set** | **MOGA** | **MALAGUTI [16]** | **Improvement** |
|---|---|---|---|
| GEOM20 | 150 | 149 | -1 |
| GEOM20a | 170 | 169 | -1 |
| GEOM20b | 44 | 44 | 0 |
| GEOM30 | 160 | 160 | 0 |
| GEOM30a | 212 | 210 | -2 |
| GEOM30b | 77 | 77 | 0 |
| GEOM40 | 167 | 167 | 0 |
| GEOM40a | 214 | 214 | 0 |
| GEOM40b | 75 | 74 | -1 |
| GEOM50 | 225 | 224 | -1 |
| GEOM50a | 326 | 316 | -10 |
| GEOM50b | 87 | 83 | -4 |
| GEOM60 | 259 | 258 | -1 |
| GEOM60a | 367 | 357 | -10 |
| GEOM60b | 119 | 115 | -4 |
| GEOM70 | 274 | 272 | -2 |
| GEOM70a | 479 | 473 | -6 |
| GEOM70b | 123 | 117 | -6 |
| GEOM80 | 389 | 388 | -1 |
| GEOM80a | 376 | 363 | -13 |
| GEOM80b | 143 | 141 | -2 |
| GEOM90 | 338 | 332 | -6 |
| GEOM90a | 387 | 382 | -5 |
| GEOM90b | 155 | 144 | -11 |
| GEOM100 | 412 | 410 | -2 |
| GEOM100a | 460 | 444 | -16 |
| GEOM100b | 172 | 156 | -16 |
| GEOM110 | 394 | 383 | -11 |
| GEOM110a | 504 | 490 | -14 |
| GEOM110b | 213 | 206 | -7 |
| GEOM120 | 411 | 396 | -15 |
| GEOM120a | 564 | 559 | -5 |
| GEOM120b | 203 | 191 | -12 |

As seen in Table 4.11, Table 4.12 and Table 4.13, some improvements are achieved compared to Prestwich [19] and Lim [15]. Among the 33 instances of GEOM benchmarks, the hybrid MOGA finds better solutions in 10 instances compared to [15]. In 10 of the instances, the same solution is found with [15]. In the remaining 13 instances, the hybrid MOGA performs worse compared to the result in [15]. The hybrid MOGA also improves 12 instances and achieves the same result on five instances compared to Prestwich [19]. However, the hybrid MOGA performance is low compared to Malaguti[16]. The hybrid MOGA achieves the same result on only five instances compared to [16]. Summary of performance comparison of these three algorithms are presented in Table 4.14.

Table 4.14. MOGA comparison with other approaches

| **Our Framework** | **Quality** | **PRESTWICH [19]** | **LIM [15]** | **MALAGUTI [16]** |
|---|---|---|---|---|
| MOGA | Better | 12 | 10 | 0 |
| | Same | 5 | 10 | 5 |
| | Worse | 16 | 13 | 28 |

# 5. CONCLUSION & FUTURE WORK

In this study, a hybrid framework is proposed to solve multi objective optimization problems. A well known multi objective optimization problem; BMCP is selected as the testbed for the proposed algorithm. Also a novel crossover operator that is hybridized with ANN is also designed in order to increase the success rate of reproduction process in this study.

Satisfactory results are obtained on the Benchmark instances utilized. Multi objective genetic algorithms (MOGAs) have not been applied to BMCP before in the literature. The multi-objective framework eliminates the need for determining the optimal bandwidth value for the problem instance. Optimization of the bandwidth value is one of the objective functions in the problem. MOGA framework makes it possible to run a parallel search on the solutions with different bandwidth values.

The hybrid search framework proposed for solving BMCP has been investigated in detail. Various tests are performed in order to determine the contribution of different operators used in the framework. This approach could be easily applied to other similar partitioning problems with minor updates.

The ANN-crossover proposed in this study is a novel operator which can increase the performance of the general framework. The operation makes it possible to breed better offspring compared to standard crossover operators. It has been observed that the performance increase is significant on the problem instances used. GAs have been used to improve the performance of ANNs in the literature, however it is a new approach to use ANNs to improve the performance of GA operators.

Location information of the color groups is the only input provided to the ANN in the current framework. ANNs are able to use this information to guide the crossover operation for better offspring. As the future work, color assignments for the vertices can also be used in the training phase of ANNs. Further enhancement could be obtained when extra information is used in the training process.

As mentioned in the Introduction, BMCP is a difficult problem that can be a model for various industrial applications. The proposed framework is also intended to be tested on real-world problem instances in the next phase of the study. The proposed framework can be considered as an infrastructure that can be used for solving different real world problems.

# REFERENCES


1. Hertz, J., "Introduction to the theory of neural computation", *Basic Books*, vol. 1, 1991.

2. Korkmaz, E., "Multi-objective genetic algorithms for grouping problems", *Applied Intelligence*, vol. 33, no. 2, pp. 179–192, 2010.

3. Ülker, Ö., Özcan, E. and Korkmaz, E., "Linear linkage encoding in grouping problems: applications on graph coloring and timetabling", *Practice and Theory of Automated Timetabling VI.,Lecture Notes in Computer Science*, vol. 3867/2007, pp. 347–363, 2007.

4. Koster A. M. C. A., *Frequency Assignment - Models and Algorithms*, Ph.D. Thesis, Maastricht University, 1999.

5. Dorne, R. and Hao, J., "Tabu search for graph coloring, tcolorings and set t-colorings" , *Meta-heuristics: Advances and trends in local search paradigms for optimization*, pp. 77–92, 1998.

6. Lim, A., Zhang, X. and Zhu, Y., "A hybrid methods for the graph coloring and its related problems", *in Proceedings of MIC2003: The Fifth Metaheuristic International Conference*, Kyoto, Japan, 2003.

7. Galinier, P. and Hao, J., "Hybrid evolutionary algorithms for graph coloring" , Journal of combinatorial optimization, vol. 3, no. 4, pp. 379–397, 1999.

8. Deb, K., "Multi-objective optimization using evolutionary algorithms", *John Wiley & Sons*, vol. 16, 2001.

9. Dorne, R. And Hao, J.K., "Tabu search for graph coloring, T-colorings and set T-colorings", *in Meta-heuristics*, pp. 77-92, Springer, 1999.

10. Brélaz, D., "New methods to color the vertices of a graph", *Communications of the ACM*, vol. 22, no. 4, pp. 251-256, 1979.

11. Łukasik, S., Kokosiński, Z. and Świętoń, G., "Parallel simulated annealing algorithm for graph coloring problem", *Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg*, pp. 229-238, 2008.

12. Trick, M., "Computational symposium: Graph coloring and its generalizations", 2002. http://mat.gsia.cmu.edu/COLOR02/

13. Culberson, J.C. and Luo, F., "Exploring the k-colorable landscape with iterated greedy", *Cliques, coloring, and satisfiability: second DIMACS implementation challenge*, vol. 26, pp. 245-284, 1996.

14. Joslin, D., and Clements, D.P., "Squeaky wheel optimization", *J. Artif. Intell. Res.(JAIR)*, vol. 10, pp. 353-373, 1999.

15. Lim, A., Zhu, Y., Louand, Q. and Rodrıgues, B. "Heuristic methods for graph coloring problems", in Proceedings of the 2005 ACM Symposium on Applied Computing, pp. 933-939, ACM, 2005.

16. Malaguti, E., Toth, P., "An evolutionary approach for bandwidth multicoloring problems", *European Journal of Operational Research*, vol. 189, no. 3, pp. 638-651, 2008.

17. Plumettaz, M., Schindl, D. and Zufferey, N., "Ant local search and its efficient adaptation to graph colouring", *Journal of the Operational Research Society*, vol. 61, no. 5, pp. 819–826, 2009.

18. Dorigo, M., and Stuetzle, T. "Handbook of metaheuristics", *In F. Glover and G. Kochenberger (Eds). Chap. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*, pp. 251–285, 2002.

19. Prestwich, S., “Generalised graph colouring by a hybrid of local search and constraint programming”, *DiscreteAppliedMathematics*, vol. 156, no. 2, pp. 148-158, 2008.

20. Raphaël, D. and Hao, J.K., "Constraint handling in evolutionary search: A case study of the frequency assignment", *Parallel Problem Solving from Nature—PPSN IV. Springer Berlin Heidelberg*, pp. 801-810, 1996.

21. Knowles, J. and Corne, D., “M-paes: A memetic algorithm for multiobjective optimization”, *in Evolutionary Computation. Proceedings of the 2000 Congress on*, vol. 1. IEEE, pp. 325–332, 2000.

22. Knowles, J. and Corne, D., "The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation", *Evolutionary Computation, CEC 99. Proceedings of the 1999 Congress on*, vol. 1. IEEE, 1999.

23. Zitzler, E. and L. Thiele, “Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach”, *IEEE Transactions on Evolutionary Computation 3*, vol. 4, pp. 257–271, 1999.

24. Zitzler, E., Laumanns, M. and Thiele, L., "SPEA2: Improving the strength Pareto evolutionary algorithm", 2001.

25. Fausett, L.V., “Fundamentals of neural networks”, *Prentice-Hall*, 1994.

26. Rumelhart, D.E., Hinton, G.E. and Williams, R.J., "Learning representations by back-propagating errors", *Cognitive modeling*, vol. 323, pp. 533-536, 1988.

27. Janson, D J. and Frenzel, J.F., "Training product unit neural networks with genetic algorithms", *IEEE Expert*, vol. 8, no. 5, pp. 26-33, 1993.

28. Montana, D.J. and Davis, L., "Training Feedforward Neural Networks Using Genetic Algorithms", *IJCAI*, vol. 89, 1989.

29. Yoon, B., Holmes, D. J., Langholz, G., and Kandel, A., “Efficient genetic algorithms for training layered feedforward neural networks”, *Information Sciences*, vol. 76, no. 1, pp. 67-85, 1994.

30. Mandischer, M., "Representation and evolution of neural networks", *Artificial Neural Nets and Genetic Algorithms. Springer Vienna*, pp. 643-649, 1993.

31. Braun, H., and Weisbrod, J., “Evolving neural feedforward networks”, *In Artificial Neural Nets and Genetic Algorithms, Springer Vienna*, pp. 25-32, 1993.

32. Braun, H., and Zagorski, P., “ENZO-II-A powerful design tool to evolve multilayer feed forward networks”, *In Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference*, pp. 278-283, 1994.

33. Yalkın, C. and Korkmaz, E., "Neural Network World: A Neural Network Based Selection Method For Genetıc Algorithms", *Neural Network World*, vol. 6, no. 12, pp. 495-510, 2012.

34. Horn, J., Nafpliotis, N. and Goldberg, D., “A niched pareto genetic algorithm for multiobjective optimization”, *in Evolutionary Computation, IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. Ieee, pp. 82–87, 1994.

35. Zitzler, E., “Evolutionary algorithms for multiobjective optimization: Methods and applications”, *Shaker*, 1999.

# APPENDIX A: DETAILS OF THE STRUCTURES USED IN THE CODE

The framework used in this thesis is coded using c programming language. Various structures are used to model the chromosomes and the search process. Details of the C structures that are used in this thesis are given in this section.

Algorithm A.1. Vertex structure

```
typedef struct VertexStruct{
        int ID;
        int weight;
        ColorDiff *selfCost;
        int colorPos;
        int edgeNum;
        struct VertexStruct **neighbors;
        ColorDiff **edgeWeights;
} Vertex;
```

Vertex structure is given in Algorithm A.1. Each vertex in the graph structure has an ID which is represented by an integer variable *ID*. In BCMP, more than a single color can be assigned to each vertex. The color amount that should be assigned to each vertex is represented with an integer variable *weight*. The variable *selfCost* determines the color difference between the colors of the same vertex. The integer variable *colorPos* holds the starting index of each vertex's color data in the color array. The *neighbor* pointer corresponds to neighbors of the each vertex. The number of the neighbors is kept in the integer variable *edgeNum*. The color difference between the each vertex and its neighbors is represented by *edgeWeights* pointer.

Algorithm A.2. Edge structure

```
typedef struct Edge{
        int c1;
        int c2;
        ColourDiff diff;
        int n1;
        int n2;
} Edge;
```

A single edge exists between two adjacent vertices in a graph. However, a single vertex can have more than one color in BMCP. And all colors of adjacent vertices have to satisfy some weight constraints. Therefore, each color in a vertex is considered as a node and a separate edge connection is assumed to exist between such node pairs of adjacent vertices. Such edges are represented by the edge structure in Algorithm A.2. The integer variables *c1* and *c2* are used to represent the color index in the color array. The variables *n1* and *n2* correspond to the vertices that these colors belong to. The color difference between *c1* and *c2* of *n1* and *n2* vertices is represented by the variable *diff*.

Algorithm A.3. Problem structure

```
struct Problem{
        int vertexNum;
        int edgeNum;
        Node *nodeList;
        int totalColor;
        Edge *edgeList;
        int totalEdge;
}p;
```

The problem structure is shown in Algorithm A.3. It is used to keep track of general properties of the graph. Vertex number and edge number of the graph is kept in the *vertexNum* and *edgeNum* variables. Information of every vertex is kept in the *nodeList*

pointer. Total number of locations that is needed to be colored in the graph is kept in the integer variable *totalColor*. Information of the edges in between every color in the graph is kept with *edgeList* pointer and the total number of these edges is kept in *totalEdge* variable.

Algorithm A.4. Chromosome structure

```
typedef struct{
        int *colorData;
        int maxColor;
        int minColor;
        int conflictsNo;
        int bandwidth;
        unsigned int hash;
        int pf;
}Chromosome;
```

Chromosomes in the population are represented by the chromosome structure which is given in Algorithm A.4. The *colorData* pointer used to hold the color array. *maxColor* and *minColor* are used to keep track of maximum and minimum colors used in the chromosomes. They are also used to determine the bandwidth which is represented by the integer variable *bandwidth*. Subtracting *minColor* from *maxColor* roughly gives us the bandwidth value of the chromosome. Unsigned integer variable *hash* is used to distinguish each chromosome from each other. We do not want to have the same chromosomes in the population. The variable *pf* shows whether a chromosome is in pareto front or not. It is set to one if the chromosome is in the pareto front, otherwise it is set to zero.

Algorithm A.5. Generation structure

```
typedef struct{
        Chromosome *s;
        int bestChromosomeBandwidth;
        int bestChromosomeConflicts;
        int bestIndex;
        int pfSize;
        int restSize;
        Chromosome **paretoFront;
        Chromosome **rest;
}Generation;
```

General information related to the chromosomes is kept by using the generation structure. The generation structure is given in Algorithm A.5. The pointer *s* represents chromosomes. *bestChromosomeBandwidth* variable and *bestChromosomeConflictsare are* used to keep track of the best bandwidth and the best conflict number obtained so far in the population. The best chromosome's index in the current population is kept in the *bestIndex* variable. Number of elements in the pareto front and in the rest of the population are kept in *pfSize* and *restSize*. Every chromosome in the pareto front is kept in the *paretoFront* pointer. The chromosomes that are not in the pareto front is kept in the *rest* pointer.

# APPENDIX B: BANDWIDTH RESULTS OBTAINED BY USING THE ANN-CROSSOVER

Table B.1. GEOM20 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 20 | 151 | 151 |
| | 150 | 150 |
| | 153 | 150 |
| | 152 | 151 |
| | 153 | 150 |
| | 153 | 152 |
| | 152 | 152 |
| | 152 | 151 |
| | 152 | 151 |
| | 151 | 152 |
| 20a | 170 | 170 |
| | 170 | 170 |
| | 170 | 171 |
| | 171 | 171 |
| | 173 | 170 |
| | 170 | 170 |
| | 171 | 171 |
| | 173 | 170 |
| | 173 | 170 |
| | 173 | 170 |
| 20b | 44 | 44 |
| | 44 | 44 |
| | 44 | 44 |
| | 44 | 44 |
| | 44 | 44 |
| | 44 | 44 |
| | 44 | 44 |
| | 44 | 44 |
| | 44 | 44 |
| | 44 | 44 |

Table B.2. GEOM30 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 30 | 160 | 160 |
| | 163 | 160 |
| | 160 | 160 |
| | 161 | 161 |
| | 162 | 160 |
| | 161 | 160 |
| | 161 | 160 |
| | 161 | 160 |
| | 160 | 160 |
| | 162 | 163 |
| 30a | 217 | 214 |
| | 213 | 218 |
| | 218 | 212 |
| | 218 | 215 |
| | 215 | 212 |
| | 219 | 213 |
| | 217 | 214 |
| | 217 | 213 |
| | 217 | 216 |
| | 214 | 216 |
| 30b | 77 | 77 |
| | 78 | 78 |
| | 78 | 78 |
| | 77 | 77 |
| | 78 | 77 |
| | 78 | 78 |
| | 77 | 78 |
| | 78 | 77 |
| | 78 | 77 |
| | 78 | 78 |

Table B.3. GEOM40 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 40 | 172 | 168 |
| | 169 | 168 |
| | 171 | 168 |
| | 168 | 167 |
| | 170 | 168 |
| | 167 | 169 |
| | 169 | 168 |
| | 170 | 168 |
| | 172 | 169 |
| | 168 | 171 |
| 40a | 221 | 218 |
| | 228 | 214 |
| | 224 | 217 |
| | 222 | 221 |
| | 222 | 223 |
| | 220 | 217 |
| | 218 | 214 |
| | 221 | 217 |
| | 226 | 217 |
| | 216 | 224 |
| 40b | 75 | 76 |
| | 78 | 76 |
| | 79 | 77 |
| | 76 | 78 |
| | 76 | 75 |
| | 77 | 75 |
| | 78 | 78 |
| | 76 | 76 |
| | 79 | 78 |
| | 75 | 75 |

Table B.4. GEOM50 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 50 | 225 | 229 |
| | 229 | 228 |
| | 227 | 225 |
| | 228 | 226 |
| | 227 | 225 |
| | 225 | 227 |
| | 228 | 227 |
| | 227 | 227 |
| | 227 | 229 |
| | 228 | 225 |
| 50a | 342 | 336 |
| | 339 | 340 |
| | 342 | 337 |
| | 340 | 337 |
| | 344 | 328 |
| | 334 | 339 |
| | 337 | 336 |
| | 336 | 336 |
| | 342 | 330 |
| | 326 | 326 |
| 50b | 91 | 88 |
| | 90 | 88 |
| | 92 | 90 |
| | 94 | 90 |
| | 93 | 88 |
| | 88 | 88 |
| | 93 | 89 |
| | 91 | 87 |
| | 90 | 88 |
| | 90 | 88 |

Table B.5. GEOM60 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 60 | 264 | 261 |
| | 269 | 263 |
| | 263 | 263 |
| | 267 | 262 |
| | 265 | 259 |
| | 258 | 259 |
| | 264 | 261 |
| | 265 | 261 |
| | 261 | 265 |
| | 259 | 260 |
| 60a | 374 | 368 |
| | 378 | 374 |
| | 375 | 375 |
| | 374 | 369 |
| | 372 | 367 |
| | 378 | 371 |
| | 379 | 370 |
| | 378 | 376 |
| | 373 | 372 |
| | 379 | 371 |
| 60b | 123 | 122 |
| | 123 | 119 |
| | 118 | 121 |
| | 124 | 122 |
| | 123 | 124 |
| | 123 | 124 |
| | 121 | 120 |
| | 125 | 123 |
| | 124 | 124 |
| | 122 | 121 |

Table B.6. GEOM70 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 70 | 275 | 276 |
| | 281 | 274 |
| | 280 | 277 |
| | 285 | 276 |
| | 280 | 285 |
| | 283 | 274 |
| | 278 | 277 |
| | 280 | 277 |
| | 278 | 278 |
| | 282 | 278 |
| 70a | 493 | 483 |
| | 484 | 483 |
| | 487 | 484 |
| | 481 | 483 |
| | 480 | 485 |
| | 493 | 485 |
| | 484 | 484 |
| | 481 | 485 |
| | 485 | 479 |
| | 491 | 487 |
| 70b | 129 | 126 |
| | 127 | 128 |
| | 130 | 131 |
| | 127 | 123 |
| | 125 | 127 |
| | 127 | 126 |
| | 132 | 131 |
| | 126 | 128 |
| | 127 | 124 |
| | 127 | 130 |

Table B.7. GEOM80 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 80 | 400 | 399 |
| | 399 | 391 |
| | 396 | 395 |
| | 400 | 391 |
| | 394 | 397 |
| | 393 | 389 |
| | 401 | 391 |
| | 397 | 393 |
| | 395 | 392 |
| | 400 | 394 |
| 80a | 387 | 378 |
| | 383 | 378 |
| | 389 | 386 |
| | 383 | 391 |
| | 382 | 380 |
| | 388 | 384 |
| | 386 | 383 |
| | 385 | 381 |
| | 394 | 387 |
| | 383 | 376 |
| 80b | 146 | 147 |
| | 145 | 144 |
| | 145 | 146 |
| | 144 | 144 |
| | 145 | 145 |
| | 149 | 143 |
| | 144 | 146 |
| | 144 | 143 |
| | 141 | 144 |
| | 143 | 145 |

Table B.8. GEOM90 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 90 | 348 | 344 |
| | 344 | 343 |
| | 345 | 340 |
| | 341 | 340 |
| | 345 | 338 |
| | 341 | 345 |
| | 344 | 343 |
| | 339 | 341 |
| | 344 | 340 |
| | 346 | 341 |
| 90a | 397 | 398 |
| | 404 | 400 |
| | 392 | 391 |
| | 397 | 398 |
| | 395 | 396 |
| | 397 | 388 |
| | 393 | 387 |
| | 395 | 398 |
| | 388 | 395 |
| | 390 | 390 |
| 90b | 162 | 163 |
| | 159 | 158 |
| | 162 | 157 |
| | 157 | 155 |
| | 163 | 158 |
| | 163 | 158 |
| | 165 | 158 |
| | 160 | 160 |
| | 161 | 165 |
| | 158 | 158 |

Table B.9. GEOM100 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 100 | 425 | 427 |
| | 426 | 423 |
| | 432 | 415 |
| | 429 | 421 |
| | 429 | 432 |
| | 432 | 429 |
| | 430 | 420 |
| | 436 | 412 |
| | 425 | 422 |
| | 428 | 423 |
| 100a | 469 | 460 |
| | 464 | 463 |
| | 470 | 468 |
| | 477 | 469 |
| | 471 | 463 |
| | 476 | 463 |
| | 466 | 463 |
| | 475 | 461 |
| | 475 | 467 |
| | 470 | 468 |
| 100b | 176 | 179 |
| | 168 | 180 |
| | 180 | 175 |
| | 179 | 172 |
| | 179 | 180 |
| | 172 | 172 |
| | 178 | 175 |
| | 168 | 175 |
| | 173 | 174 |
| | 175 | 183 |

Table B.10. GEOM110 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 110 | 392 | 396 |
| | 394 | 397 |
| | 403 | 394 |
| | 406 | 396 |
| | 403 | 406 |
| | 406 | 396 |
| | 392 | 403 |
| | 402 | 399 |
| | 406 | 395 |
| | 409 | 403 |
| 110a | 517 | 512 |
| | 523 | 513 |
| | 516 | 506 |
| | 525 | 512 |
| | 526 | 504 |
| | 531 | 512 |
| | 513 | 506 |
| | 518 | 507 |
| | 511 | 511 |
| | 514 | 518 |
| 110b | 217 | 215 |
| | 218 | 214 |
| | 215 | 217 |
| | 225 | 218 |
| | 215 | 216 |
| | 218 | 218 |
| | 222 | 215 |
| | 219 | 214 |
| | 216 | 217 |
| | 219 | 213 |

Table B.11. GEOM120 results

| Benchmark Problem Set | Max Group Crossover | ANN Crossover |
|---|---|---|
| 120 | 418 | 415 |
| | 425 | 423 |
| | 422 | 411 |
| | 424 | 420 |
| | 414 | 423 |
| | 429 | 416 |
| | 416 | 422 |
| | 421 | 419 |
| | 423 | 418 |
| | 427 | 416 |
| 120a | 576 | 576 |
| | 590 | 577 |
| | 586 | 566 |
| | 581 | 580 |
| | 593 | 573 |
| | 590 | 575 |
| | 569 | 576 |
| | 576 | 564 |
| | 585 | 576 |
| | 594 | 570 |
| 120b | 213 | 214 |
| | 209 | 208 |
| | 213 | 208 |
| | 210 | 211 |
| | 209 | 209 |
| | 216 | 206 |
| | 208 | 209 |
| | 210 | 204 |
| | 210 | 203 |
| | 209 | 206 |