

LINEAR LINKAGE ENCODING IN GENETIC ALGORITHMS

by  
Özgür ÜLKER

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  
2006

## LINEAR LINKAGE ENCODING IN GENETIC ALGORITHMS

## ACKNOWLEDGEMENTS


I would like to thank my advisors, Assist. Prof. Dr. Ender Özcan and Dr. Emin Erkan Korkmaz for their guidance and support during the preparation of this M. Sc. thesis.

APPROVED BY:

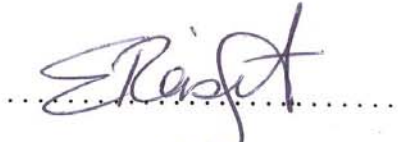
Assist. Prof. Dr. Ender Özcan  
(Thesis Supervisor)



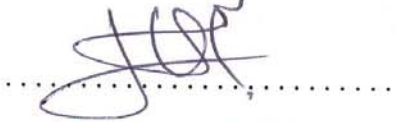
Assist. Prof. Dr. Emin Erkan Korkmaz  
(Thesis Co-supervisor)



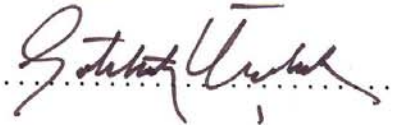
Assoc. Prof. Dr. Raşit Eskicioğlu



Prof. Dr. Ahmet Coşkun Sönmez



Assoc. Prof. Dr. Göktürk Üçoluk



DATE OF APPROVAL: 27/07/2006

## ACKNOWLEDGEMENTS

I want to thank my advisors Dr. Ender Özcan and Dr. Emin Erkan Korkmaz for their guidance and valuable feedback throughout this M. Sc. study.

This research is funded by TUBITAK (The Scientific and Technological Research Council of Turkey) under the grant number 105E027. I am grateful for their financial support.

Last but not the least, I want to thank my family for their support during this thesis work.

## **ABSTRACT**

### **LINEAR LINKAGE ENCODING IN GENETIC ALGORITHMS**

Linear Linkage Encoding (LLE) is a recently proposed representation scheme for evolutionary algorithms (EA). This representation has been previously used to solve data clustering problem. It is also suitable for other grouping problems. In this thesis, this new encoding scheme is investigated on two other grouping problems; graph coloring and bin packing. The main focus of this work is to investigate genetic operators suitable for LLE. Performance enhancing crossover operators for graph coloring problem based on LLE are proposed and compared to the existing ones. Traditional crossover operators with additional placement heuristics are tested on bin packing problem. Initial results show that Linear Linkage Encoding is a viable candidate for grouping problems whenever appropriate genetic operators are chosen.

## ÖZET

# GENETİK ALGORİTMALARDA DOĞRUSAL BAĞLANTI GÖSTERİMİ

Doğrusal Bağlantı Gösterimi (DBG) evrimsel algoritmalar (EA) için yakın zamanda önerilen bir gösterim şeklidir. Bu gösterim daha önce veri kümeleme probleminin çözümünde kullanılmıştır. Diğer gruplama problemleri için de uygundur. Bu tezde, bu yeni gösterim şekli iki gruplama problemi üzerinde incelenmiştir. Bu çalışmanın ana odak noktası DBG ile kullanılabilir genetik uzmanların incelenmesidir. Çizge boyama problemi için DBG ile uyumlu başarımları artırıcı çaprazlama uzmanları önerilmiş ve var olanlar ile karşılaştırılmıştır. Geleneksel çaprazlama uzmanları, yerleştirme buluşsal yöntemleriyle beraber kutu paketleme problemi üzerinde denenmiştir. İlk sonuçlar doğrusal bağlantı gösteriminin uygun genetik uzmanlar kullanıldığında gruplama problemleri için başarılı bir aday olduğunu göstermiştir.

## TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . .	iii
ABSTRACT . . . . .	iv
ÖZET . . . . .	v
LIST OF FIGURES . . . . .	ix
LIST OF TABLES . . . . .	xi
LIST OF SYMBOLS/ABBREVIATIONS . . . . .	xiv
1. INTRODUCTION . . . . .	1
2. LITERATURE SURVEY . . . . .	4
2.1. Graph Coloring Problem . . . . .	4
2.1.1. Application Areas of Graph Coloring . . . . .	4
2.1.2. Approaches for Solving Graph Coloring Problem . . . . .	5
2.1.2.1. DSATUR . . . . .	6
2.1.2.2. RLF . . . . .	6
2.1.2.3. Genetic Algorithms . . . . .	6
2.1.2.4. Local Search . . . . .	7
2.1.2.5. Tabu Search . . . . .	7
2.1.2.6. Simulated Annealing . . . . .	8
2.1.2.7. Hybrid Algorithms . . . . .	9
2.1.2.8. Variable Neighborhood Search . . . . .	9
2.1.3. Exam Timetabling as a Grouping Problem . . . . .	10
2.2. Bin Packing Problem . . . . .	10
2.2.1. Application Areas of Bin Packing . . . . .	11
2.2.2. Approaches for solving Bin Packing Problem . . . . .	11
2.2.2.1. First Fit . . . . .	11
2.2.2.2. Best Fit . . . . .	12
2.2.2.3. Worst Fit . . . . .	12
2.2.2.4. Reduction Algorithm of Martello and Toth . . . . .	12
2.2.2.5. Hybrid Grouping Genetic Algorithm for Bin Packing Problem . . . . .	13

3. GENETIC ALGORITHMS . . . . .	14
3.1. Representation . . . . .	14
3.2. Selection . . . . .	15
3.2.1. Roulette Wheel Selection . . . . .	15
3.2.2. Rank Based Selection . . . . .	15
3.2.3. Tournament Based Selection . . . . .	17
3.3. Crossover . . . . .	17
3.4. Mutation . . . . .	18
3.5. Replacement . . . . .	19
3.6. Memetic Algorithms . . . . .	19
3.7. Multi-Objective Genetic Algorithms . . . . .	20
3.7.1. Implementations of Multi-objective Genetic Algorithms . . . . .	22
3.7.1.1. Non-Dominated Sorting Genetic Algorithm (NSGA) . . . . .	22
3.7.1.2. Multi Objective Genetic Algorithm (MOGA) . . . . .	22
3.7.1.3. Niched Pareto Genetic Algorithm (NPGA) . . . . .	23
4. LINEAR LINKAGE ENCODING FOR GROUPING PROBLEMS . . . . .	25
4.1. Previous Representations in Grouping Problems . . . . .	25
4.1.1. Number Encoding . . . . .	25
4.1.2. Group Encoding . . . . .	26
4.1.3. Hybrid Grouping Genetic Algorithm representation . . . . .	26
4.2. LLE Implementation . . . . .	26
4.3. Repair Procedure for Linear Linkage Encoding . . . . .	29
4.4. Initialization . . . . .	30
4.5. Crossover and Mutation . . . . .	31
5. A MULTI-OBJECTIVE GENETIC ALGORITHM FOR GRAPH COLORING AND TIMETABLING . . . . .	32
5.1. Initialization . . . . .	32
5.2. Selection . . . . .	33
5.3. Objective Functions . . . . .	34
5.4. Redundancy and Genetic Operators . . . . .	34
5.4.1. Cardinality Based Ordering . . . . .	35
5.4.2. Lowest Index Ordering . . . . .	35

5.5. Crossover . . . . .	36
5.5.1. Greedy Partition Crossover . . . . .	36
5.5.2. Lowest Index First Crossover . . . . .	38
5.5.3. Lowest Index Max Crossover . . . . .	40
5.6. Mutation . . . . .	41
5.7. Replacement . . . . .	41
5.8. Experimental Results . . . . .	42
6. A HYBRID GENETIC ALGORITHM FOR BIN PACKING . . . . .	50
6.1. Initialization . . . . .	50
6.2. Fitness Function . . . . .	50
6.3. Selection . . . . .	51
6.4. Crossover . . . . .	51
6.5. First Fit Decreasing Rearrangement . . . . .	52
6.6. Mutation . . . . .	53
6.7. Replacement . . . . .	53
6.8. Experimental Results . . . . .	53
7. CONCLUSIONS . . . . .	61
APPENDIX A: EXPERIMENTAL RESULTS - GRAPH COLORING . . . . .	63
APPENDIX B: EXPERIMENTAL RESULTS - BIN PACKING . . . . .	67
REFERENCES . . . . .	73



## LIST OF FIGURES

Figure 3.1.	Pseudocode of a genetic algorithm . . . . .	14
Figure 3.2.	a) Binary representation b) Floating point representation c) Order based representation. . . . .	15
Figure 3.3.	Selection probabilities for different selection methods a) Roulette wheel selection b) Ranking based selection c) Tournament selection with <i>toursize</i> = 2. . . . .	16
Figure 3.4.	a) One point crossover b) N=2 point crossover c) uniform Crossover. .	18
Figure 3.5.	Selection algorithm for niched pareto genetic algorithm . . . . .	24
Figure 4.1.	LLE array and LOP graphs . . . . .	27
Figure 4.2.	Repair procedure for linear linkage encoding . . . . .	30
Figure 4.3.	Grafting mutation in linear linkage encoding . . . . .	31
Figure 5.1.	Pseudocode of the greedy partition crossover . . . . .	37
Figure 5.2.	a) Two Parents in LLE array and LOP graph form. b) Resulting offspring from greedy partition crossover - lowest index ordering c) Resulting offspring from greedy partition crossover - cardinality based ordering. d) Resulting offspring from lowest index first crossover. e) Resulting offspring from lowest index max crossover. . . . .	38
Figure 5.3.	Pseudocode of the lowest index first crossover . . . . .	39

Figure 5.4.	Pseudocode of the lowest index max crossover . . . . .	40
Figure 5.5.	Average number of groups for instances in DIMACS benchmark. . . . .	45
Figure 5.6.	Average number of groups obtained for instances in Carter's Benchmark. . . . .	48
Figure 6.1.	Pseudocode of the modified uniform crossover . . . . .	51

## LIST OF TABLES

Table 5.1.	Test setup for graph coloring instances . . . . .	42
Table 5.2.	Data characteristics of the problem instances from the DIMACS suite .	43
Table 5.3.	Best colorings obtained for the instances in the DIMACS benchmark suite	46
Table 5.4.	Data characteristics of the problem instances from the Carter bench- mark suite . . . . .	47
Table 5.5.	Best colorings obtained for the instances in the Carter’s benchmark suite	48
Table 6.1.	Test setup for bin packing instances . . . . .	54
Table 6.2.	Rankings for first fit (decreasing) - 1 bin deleted . . . . .	55
Table 6.3.	Rankings for first fit (decreasing) - 2 bins deleted . . . . .	56
Table 6.4.	Rankings for first fit (decreasing) - 4 bins deleted . . . . .	56
Table 6.5.	Rankings for hill climbers in one point crossover . . . . .	58
Table 6.6.	Rankings for hill climbers in lowest index max crossover . . . . .	58
Table 6.7.	Rankings for hill climbers in uniform crossover . . . . .	59
Table 6.8.	Rankings for hill climbers in modified uniform crossover . . . . .	59
Table 6.9.	Mean number of bins obtained in best setups for each crossover . . . .	60

Table A.1.	Average best, standard deviation and best results for the Carter's benchmark instances - Results for lowest index maximum and lowest index first crossovers . . . . .	63
Table A.2.	Average best, standard deviation and best results for the Carter's benchmark instances - Results for greedy partition lowest index and cardinality based crossovers . . . . .	64
Table A.3.	Average best, standard deviation and best results for the DIMACS Benchmark instances - Results for lowest index maximum and lowest index first crossovers . . . . .	65
Table A.4.	Average best, standard deviation and best results for the DIMACS benchmark instances - Results for greedy partition lowest index and cardinality based crossovers . . . . .	66
Table B.1.	Best packings obtained for the instances in the Falkenauer's benchmark suite - First fit decreasing with 1 bin deleted . . . . .	67
Table B.2.	Best packings obtained for the instances in the Falkenauer's benchmark suite - First fit decreasing with 2 bins deleted . . . . .	68
Table B.3.	Best packings obtained for the instances in the Falkenauer's benchmark suite - First fit decreasing with 4 bins deleted . . . . .	69
Table B.4.	Best packings obtained for the instances in the Falkenauer's benchmark suite - First fit with 1 bin deleted . . . . .	70
Table B.5.	Best packings obtained for the instances in the Falkenauer's benchmark suite - First fit with 2 bins deleted . . . . .	71

Table B.6. Best packings obtained for the instances in the Falkenauer’s benchmark suite - First fit with 4 bins deleted . . . . . 72

## LIST OF SYMBOLS/ABBREVIATIONS

$\chi(G)$	Chromatic Number
1PTX	One Point Crossover
BPP	Bin Packing Problem
CB	Cardinality Based
DIMACS	Discrete Mathematics and Theoretical Computer Science Center
DSATUR	Degree of Saturation
EA	Evolutionary Algorithm
FF	First Fit
FFD	First Fit Decreasing
GA	Genetic Algorithm
GCP	Graph Coloring Problem
GE	Group Encoding
GPX	Greedy Partition Crossover
HGGA	Hybrid Grouping Genetic Algorithm
LI	Lowest Index
LIFX	Lowest Index First Crossover
LIMX	Lowest Index Max Crossover
LLE	Linear Linkage Encoding
LOP	Labeled Oriented Pseudo
MA	Memetic Algorithm
MOGA	Multi-Objective Genetic Algorithm
MUX	Modified Uniform Crossover
NE	Number Encoding
NP	Non-Deterministic Polynomial
NPGA	Niched Pareto Genetic Algorithm
NSGA	Non-Dominated Sorting Genetic Algorithm
RLF	Recursive Largest First
UX	Uniform Crossover

## 1. INTRODUCTION

Grouping problems [1] are generally concerned with partitioning a set  $V$  of items into a collection of mutually disjoint subsets  $V_i$  of  $V$  such that

$$V = V_1 \cup V_2 \cup V_3 \dots \cup V_N \quad \text{and} \quad V_i \cap V_j = \emptyset \quad \text{where} \quad i \neq j. \quad (1.1)$$

Obviously, the aim of these problems is to partition the members of set  $V$  into  $N$  different groups ( $1 \leq N \leq |V|$ ) where each item is in exactly one group. In most of the grouping problems, not all possible groupings are permitted; a valid solution usually has to comply with a set of constraints. For example in graph coloring, the vertices in the same group must not be adjacent in the graph. In bin packing problem, the sum of the sizes of items of any group should not exceed the capacity of the bin, etc. Hence, the objective of grouping is to optimize a cost function defined over a set of valid groupings. In both graph coloring and bin packing, the objective is to minimize the number of groups (independent sets and bins respectively) subject to the aforementioned constraints.

The essential issue about grouping problems is that the objective function to be optimized is defined on a set of all valid groupings and depends on the composition of the groups of the items [2]. Obviously, a composition of a group has a natural meaning in a grouping problem which should be taken into account in an optimization method. An item without considering its group has little or no meaning during the search process. Therefore, groups or group segments should be preserved during search.

Some well known grouping problems are as follows:

- Graph Coloring: Partition an undirected graph  $G$  into a minimal number of disjoint sets such that no two vertices in a set is adjacent to each other.
- Bin Packing: Place a set of items with different sizes into a minimal number of bins such that the size of the items in each bin does not exceed the bin capacity.

- Workshop Layouting: Minimize the total intercell traffic in a group while the number of machines does not exceed the capacity of the group.
- Equal Piles: Partition a set of  $N$  numbers into  $K$  disjoint subsets such that the sum of the numbers in each set is as nearly equal as the sums of the other sets.

Various metaheuristics such as simulated annealing [3], tabu search [4], and genetic algorithms [5], have been applied to solve grouping problems such as above. A genetic algorithm (GA) [5] is a search method for solving optimization problems. The operators of GAs are inspired from the theory of Darwinian evolution [6]. In spite of the satisfactory performance of the traditional genetic algorithms on many NP optimization problems, the same achievement is not usually observed on grouping problems. This is because many evolutionary algorithms do not address the dynamics of a grouping problem: how to handle groups. The commonly used representations usually suffer from redundancies due to the ordering of groups. Moreover, the genetic material representing groups might easily be disrupted by the genetic operators and/or by the rectification process after the operators are applied. Therefore a genetic algorithm requires special operators for solving a grouping problems.

In this thesis, Linear Linkage Encoding (LLE) [7] for grouping problems is investigated. LLE is used as a representation scheme within genetic algorithms. Two well known grouping problems, graph coloring and bin packing are chosen as testbeds. LLE uses a link-based structure for objects within the same group. Genetic operators work on the encodings by altering the links. LLE previously has only been tested on small clustering problem instances [7], and it is observed that LLE performance is superior to Number Encoding (NE), the most common encoding scheme used in grouping problems. Unlike NE, LLE does not require an explicit bound on the number of groups that can be represented in a fixed-length encoding. The greatest strength of LLE is that the search space is reduced considerably. There is a one to one correspondence between the encodings and the solutions when LLE is used.

In this thesis, the potential of the LLE representation on grouping problems is presented. Previous studies denote that traditional crossover operators do not perform well. A



set of new crossover operators suitable for LLE are tested on a set of problem instances including Carter's Benchmark [8] and DIMACS Challenge Suite [9] for graph coloring problems. These crossovers are inspired from a powerful hybrid graph coloring algorithm of Galinier and Hao [10] and utilize redundancy lowering principles [11]. For the bin packing problem, traditional genetic operators with additional heuristics are tested on a set of instances provided by [12]. The proposed graph coloring crossovers were not particularly suitable due to the nature of the test instances in the bin packing. Although both graph coloring and bin packing are grouping problems, they pose different difficulties on handling groups due to epistasis, importance of specific groups and/or group sizes which may require different strategies. The main goal of this thesis is to obtain competitive results using genetic algorithms with LLE on grouping problems by addressing these points where traditional genetic algorithms are likely to fail.

This thesis is organized as follows: Chapter 1 gives a general overview of this work. Chapter 2 presents a literature survey about the main research topics of this study. The relevant literature of graph coloring and bin packing problems is provided. Main aspects of genetic algorithms are described in Chapter 3. The main research topic of this thesis, Linear Linkage Encoding is described in Chapter 4. Previous applications of LLE, implementational issues, advantages of LLE are presented in this chapter as well. Chapter 5 presents a multi-objective genetic algorithm for solving graph coloring problem using LLE. Details of new crossover operators for LLE in graph coloring are described and experimental results are provided in this chapter. Chapter 6 deals with a memetic genetic algorithm for solving bin packing problem by using traditional operators on LLE. Finally, conclusions and further research directions are provided in Chapter 7. Furthermore, Appendix A presents additional experimental results for graph coloring algorithm, whereas additional experimental results for bin packing are presented in Appendix B.

## 2. LITERATURE SURVEY

### 2.1. Graph Coloring Problem

Graph coloring problem (GCP) is a well known combinatorial optimization problem which is proved to be NP Complete [13]. Informally stated, graph coloring is assigning colors to each vertex of an undirected graph such that no adjacent vertices should receive the same color. The minimal number of colors that can be used for a valid coloring is called the chromatic number. A more formal definition is as follows:

Given a graph  $G = (V, E)$  with vertex set  $V$  and edge set  $E$ , and given an integer  $k$ , a  $k$ -coloring of  $G$  is a function  $c : V \rightarrow 1, \dots, k$ . The value  $c(x)$  of a vertex  $x$  is called the color of  $x$ . The vertices with color  $r$  ( $1 \leq r \leq k$ ) define a color class, denoted  $V_r$ . If two adjacent vertices  $x$  and  $y$  have the same color  $r$ ,  $x$  and  $y$  are conflicting vertices, and the edge  $(x, y)$  is called a conflicting edge. If there is no conflicting edge, then the color classes are all independent sets and the  $k$ -coloring is valid. The Graph Coloring Problem is to determine the minimum integer  $k$  (the chromatic number of  $G - \chi(G)$ ) such that there exists a legal  $k$ -coloring of  $G$  [14].

#### 2.1.1. Application Areas of Graph Coloring

Graph coloring problem has many application areas [15]. Some of the most important of these are as follows:

Many scheduling problems impose for a number of pairwise restrictions on which jobs can be done concurrently. For example, in attempting to schedule courses in a university, two courses taken by a group of students should not be assigned to the same time slot. Similarly, two courses taught by an instructor should not be scheduled in the same time slot. The problem of finding the minimum number of time slots required subject to these restrictions reduces to a graph coloring problem. A detailed survey of timetabling methods can be found in [16].

Gamst [17] examines a problem in assigning frequencies to mobile radios and other users of the electromagnetic spectrum. Two customers who are in close vicinity to each other must not be assigned to the same frequency whereas distant customers can share the same frequency. The problem of minimizing the number of frequencies is again a graph coloring problem.

Register allocation is a very active application area for graph coloring. The register allocation problem is to assign variables to a limited number of hardware registers. Usually, the number of variables is far greater than the number of registers thus it becomes necessary to assign multiple variables to a single register. Variables are in conflict with each other if one is used both before and after the other within a short period of time (such as, within a subroutine). The goal is to assign variables that do not conflict while minimizing the use of system memory. Representing variables with vertices and conflicts between variables with edges register allocation reduces to graph coloring. Detailed information on register allocation can be found in Chaitin [18], Briggs et. al [19], Chow and Hennessy [20].

Garey, Johnson, and So [21] used a graph coloring algorithm for testing printed circuit boards for unintended short circuits (caused by stray lines of solder). The nets on the board corresponds to vertices in a graph and potential for a short circuit between corresponding nets can be represented with edges. Partitioning the nets into supernets where the nets in each supernet can be tested for short circuits against all other nets can be reduced to coloring.

### **2.1.2. Approaches for Solving Graph Coloring Problem**

In the literature there are many solution methods devised for finding chromatic number and solving k-coloring problems. Naturally many graph instances were produced to compare these algorithms. Most important of these benchmark instances is DIMACS Challenge Suit [9].

Early applications of GCP solvers are simple greedy constructive methods which build feasible solutions of the form  $s(x) = (x_1, x_2, \dots, x_n)$  by inserting a component  $x_i$  repeatedly

into the current partial solution

$$s(i - 1) = (x_1, x_2, \dots, x_{i-1}) \quad (2.1)$$

which initially is an empty solution  $s(0)$ . Two well known greedy heuristics for graph coloring [22] are DSATUR and RLF. Furthermore, metaheuristics such as genetic algorithms, local search, simulated annealing, tabu search and their hybrids have also been used for solving graph coloring problem.

2.1.2.1. DSATUR. The algorithm DSATUR (Degree of Saturation) of Brelaz [23] is a sequential algorithm with an ordering of the vertices which are established dynamically. Let  $F$  be a partial coloring of the vertices of the  $G$ . The degree of saturation of a vertex  $v$ ,  $deg_s(x)$  then becomes the number of different colors at the vertices adjacent to vertex  $v$ . DSATUR operates by sequentially assigning increasing color numbers to a vertex  $x$  of maximal degree  $deg_s(x)$ . The complexity of DSATUR is  $O(|V|^3)$ .

2.1.2.2. RLF. Another well known greedy heuristic Recursive Largest First (RLF) is by Leighton [24] with a worse-case time complexity of  $O(|V|^3)$ . Let  $x$  be a fixed vertex with maximal degree. Non neighbors  $y$  with a maximal number of common neighbors with vertex  $v$  are then contracted into  $x$  until  $x$  is adjacent to every other vertex. For an effective implementation,  $x$  is removed and a new vertex with maximal degree in the remaining graph is chosen. A new color class is built from vertex  $x$  and all vertices contracted into it. If  $x$  has a non-neighbor vertex yet no non-adjacent vertex with a common neighbor, then graph is unconnected so  $x$  is adjacent to all other vertices of its own component. To enforce RLF principle, the color class of  $x$  must be constructed with a vertex of maximal degree in another component.

2.1.2.3. Genetic Algorithms. Davis [25] proposed a coding as an ordering of vertices which could be used in a genetic algorithm [5]. Davis' algorithm was designed to maximize the total weights of the vertices in the graph colored with a fixed amount of colors. Unfortunately genetic algorithms give very poor results in an order based coding. No other work using pure

genetic algorithms are reported since then. It is believed that pure genetic algorithms are not competitive for this problem. There are mainly two reasons for the unsuccessful attempts of for using pure genetic implementations on graph coloring: The redundancies inherent in the representations used for the encoding of the chromosome, and lack of a suitable crossover operator which transmits the building blocks preferably with some domain knowledge.

2.1.2.4. Local Search. Local search is an optimization method in which neighbor solutions are visited in an iterative manner based on a neighborhood relation. Local search methods such as tabu search [26] and simulated annealing [27] were also applied to the graph coloring problem. Local search methods for GCP usually start with an infeasible color assignment and iteratively move to neighboring solutions to reduce the number of conflicts. Local search operators iteratively try to repair the current color assignment guided by an evaluation function. A feasible coloring is obtained when a candidate solution with zero conflicting edges is encountered.

2.1.2.5. Tabu Search. Tabu search [4] is a local search method in which the decision to move to a neighboring solution depends on short or long term memory structures. Hertz and de Werra [26] presented the first tabu search implementation which outperforms another local search method, simulated annealing on random dense graph instances. This methods attempts to find a valid *k-coloring* by partitioning the set of vertices  $V$  into  $k$  subsets. A solution  $s = \{V_1, V_2, \dots, V_k\}$  is a partition of the set of vertices  $V$  into  $k$  subsets where

$$E(V_i) = \{(v, w) \in E | v \in V_i, w \in V_i\} \quad (2.2)$$

The quality of the solution  $s$  is determined by the objective function

$$f(s) = \sum_{i=1}^k |E(V_i)| \quad (2.3)$$

A neighbor  $s'$  is generated by randomly choosing  $v' = v$  or  $w$ , such that

$$(v, w) \in E(V_1) \cup \dots \cup E(V_k)$$

and then randomly choosing a color  $j \neq i$ . The new solution is obtained by

$$V'_j = V_j \cup \{v'\} \quad (2.4)$$

$$V'_i = V_i - \{v'\} \quad (2.5)$$

$$V'_r = V_r \quad \text{for } r = 1, \dots, k \quad \text{and } r \neq i, j \quad (2.6)$$

The tabu search method selects the best neighbors  $s'$  from a randomly generated neighbor pool whose size is determined empirically. After a move of vertex  $v \in V_i$  to  $V_j$  a tabu rule forbids the move which returns  $v$  to  $V_i$  for a number of iterations (tabu tenure). A more advanced tabu search procedure employing greedy construction, configuration re-generation, dynamic tabu tenure management and proper coloring search is presented by Dorne and Hao [28].

2.1.2.6. Simulated Annealing. In simulated annealing (SA) [3] method, each point  $s$  in the search space is compared to a state of some physical system, and the function  $E(s)$  to be minimized is defined as the internal energy of the system in that state. Therefore the aim is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy. At each step SA determines some neighbors of the current state  $s$ , and probabilistically decides between moving the system to a new state  $s'$  or not. The probabilities are chosen so that the system ultimately tends to move to states of lower energy [29].

Johnson et al. [27] present three simulated annealing implementations based on three neighboring approaches: penalty-function approach (based on RLF), Kempe chain approach (a specific move which causes a major change in neighboring structure while retaining the cost function and fixed-k approach (which attempts to minimize the number of monochromatic edges in a not necessarily-legal coloring with a fixed number of color classes).

2.1.2.7. Hybrid Algorithms. Currently, hybridizations of local search and evolutionary methods constitute state of the art graph coloring algorithms. Fleurent and Ferland's genetic tabu algorithm [30] uses an efficient pre-processing technique of [26] which reduces the initial graph by removing a large number of independent sets and which colors the residual graph with coloring algorithms. However they reported that although crossovers can improve on the performance of local search, this happens only for a few graphs with high computational requirements.

In the graph coloring literature, Galinier's and Hao's [10] method of hybridization of genetic algorithms and local search is usually considered the best solution for difficult graph instances. In their method, a new offspring created by a crossover operator which transmits large independent sets to next generations is improved by a long tabu search procedure similar to Hertz and de Werra [26]. Their rationale was that coloring should be a partition of vertices not an assignment of colors to vertices, and a crossover should transmit subsets of color classes from parents to children. They used a rather small population with 5 to 10 individuals and reported that long tabu search procedure after the crossover preserves the diversity in the population. However in [31] it is reported that the tabu search procedure can be replaced with a simple descent method without affecting the performance of the algorithm.

2.1.2.8. Variable Neighborhood Search. Recently, Avanthay et al. [14] proposed a variable neighborhood search algorithm (VNS) for graph coloring problem. Let  $N^{(t)}(t = 1, \dots, t_{max})$  denote a finite set of neighborhoods where  $N^{(t)}(s)$  is the set of solutions in the  $t^{th}$  neighborhood of  $s$ . VNS tries to avoid being trapped in local minima by using multiple neighborhoods that will be used within a local search. The proposed method utilizes three operator types of neighborhoods:

- vertex neighborhoods which change the color of some conflicting vertices.
- class neighborhoods which change the color of some or all vertices of a conflicting color.
- non-increasing neighborhoods which change the color of some vertices without increasing the total number of conflicting edges.

The authors have commented that their algorithm although strong by itself is not competitive with Galinier and Hao's hybrid algorithm.

### 2.1.3. Exam Timetabling as a Grouping Problem

Exam timetabling requires satisfactory assignment of timetable slots (periods) to a set of exams. Each exam is taken by a number of students, based on a set of constraints. In most of the studies, NE like representations are used. In [32], a randomly selected light or a heavy mutation followed by a hill climbing method was applied. Various combinations of constraint satisfaction techniques with genetic algorithms can be found in [33]. Paquete et al. [34] applied a multi-objective evolutionary algorithm based on pareto ranking with two objectives: minimize the number of conflicts within the same group and between groups. Wong et al. [35] applied a GA with a non-elitist replacement strategy. After genetic operators are applied, violations are repaired with a hill climbing fixing process. In their experiments a single problem instance was used. Ozcan et. al. [36] proposed a memetic algorithm (MA) for solving exam timetabling. MA utilizes a violation directed adaptive hill climber.

Considering the task of minimizing the number of exam periods and removing the clashes, exam timetabling reduces to the graph coloring problem [24].

## 2.2. Bin Packing Problem

Bin packing problem (BPP) is a combinatorial NP hard problem in which items of different sizes has to be packed into a minimal number of bins of fixed capacity. Bin packing has many variants with respect to the number of dimensions used, the arrival of bins, and the distribution of the size of the items. In this study, only one-dimensioal bin packing problem will be considered.

In classical one-dimensional bin packing problem [37], a sequence of  $L = (a_1, a_2, \dots, a_n)$  of items, each with a size  $s(a_i) \in (0, 1]$  are packed into a minimum number of unit-capacity bins (partition them into a minimum number  $m$  of subsets  $B_1, B_2, \dots, B_m$  such that  $\sum_{a_i \in B_j} \leq 1, 1 \leq j \leq m$ ).



### 2.2.1. Application Areas of Bin Packing

One common application of bin packing is memory allocation. In a typical memory allocation system, memory is divided into fixed sized pages and assuming the size of a memory request is less than the size of a page, allocating minimal number of pages to a finite set of memory requests reduces to a bin packing problem. Reviews of memory allocation can be found in [38] and [39].

In machine scheduling [40], the task is to allocate a minimum number of identical machines to a collection of independent jobs (that is each job can be done in any machine in any order). Assuming a job  $j_i$  takes a time of  $t_i$  and the maximum time allowed is  $T$ , machine scheduling reduces to bin packing by substituting  $t_i$  and  $T$  to item size  $s_i$  and bin capacity  $C$  respectively.

A well known NP Complete problem in computer science is multiprocessor scheduling. In multiprocessor scheduling, the task is to schedule each job  $j_i$  with a length  $l_i$  to a number of processors such that the required time is minimized. Coffman et al. [41] used a bin packing algorithm to solve this problem.

Another problem which is frequently encountered in industrial applications is stock cutting [42]. In stock cutting the task is to cut a number of items (wood, roll of paper, sheet of textile) from a fixed size raw material block such that the total waste is minimized. Bin packing and stock cutting are synonymous problems which are usually differentiated by the distribution of items in their instances.

### 2.2.2. Approaches for solving Bin Packing Problem

2.2.2.1. First Fit. In first fit heuristic, an item  $a_i$  is placed in the first (lowest indexed) partially-filled bin  $B_j$  into which it will fit ( $capacity(B_j) + s(a_i) \leq 1$ ). If this is not possible a new bin with  $a_i$  as the first item is created. A variant of first fit is first fit decreasing (FFD) in which items are first sorted in decreasing weight and then items are picked up one by one beginning with the largest item and placed into bins according to the first bin that can

accomodate it.

2.2.2.2. Best Fit. In best fit heuristic, an item  $a_i$  is placed in the partially filled bin  $B_j$  with the *highest* level  $level(B_j) \leq 1 - s(a_i)$  and ties if any are broken in favor of lower index bins. Similar to FF, Best fit has a decreasing variant, in which items are again sorted in decreasing order and placed into the best-filled bin that can accomodate it. Although best fit decreasing is slightly more complicated than FFD, surprisingly it cannot beat FFD with both having the worst case performance of  $\frac{11}{9}Opt + 4$  where  $Opt$  is the number of bins in the optimal solution. [37].

2.2.2.3. Worst Fit. In worst fit heuristic, an item  $a_i$  is placed in the partially filled bin  $B_j$  with the *lowest* level  $level(B_j) \leq 1 - s(a_i)$  and ties if any are broken in favor of lower index bins. Unfortunately, it is not possible to close down a bin. However a slight modification as in almost worst fit makes it worst case performance as good as first fit or best fist. In almost worst fit an item  $a_i$  is placed in the partially filled bin with the second lowest level unless there is only one bin into which it fits, in which case it goes into that bin. If there isn't any partially filled bin  $a_i$  can fit,  $a_i$  forms a new bin.

2.2.2.4. Reduction Algorithm of Martello and Toth. Martello and Toth's branch-and-bound reduction algorithm (MTP) [43] is the basic reference in most comparative studies of bin packing. Although slow (for large instances), MTP generally gives excellent results. The MTP is based on the following dominance criterion:

A feasible set of items is defined as any subset  $F \subseteq N$  such that  $\sum_{i \in F} w_i \leq C$ . With two feasible sets  $F_1$  and  $F_2$ ,  $F_1$  dominates  $F_2$  if and only if the the number of bins in some optimal solution by setting  $B_1 = F_1$  is not greater than by setting  $B_1 = F_2$ . There exists a partition  $P_1, \dots, P_l$  of  $F_2$  and a subset  $i_1, \dots, i_l \subset F_1$  such that  $w_{i_h} \geq \sum_{k \in P_h} w_k$  for  $h = 1, \dots, l$ .

This results in a important conclusion that a solution containing  $F_1$  will not have more bins than a solution containing  $F_2$ . The MTP procedure tries to find bins dominating all

others. After such a bin is found, the problem is reduced by removing the dominating bin. In order to prevent an exponential search, in MTP only dominating bins of at most three items are taken into account.

2.2.2.5. Hybrid Grouping Genetic Algorithm for Bin Packing Problem. Falkenauer [12] uses a Hybrid Grouping Genetic Algorithm (HGGA) which is heavily modified to suit the structure of the grouping problems. His genetic algorithm works with whole bins rather than with individual items. In HGGA representation, a standard chromosome representing the ids of the items are augmented with a group part, encoding the groups on a one gene for one group basis. The important point with the genetic operators is that it works on the group part of the chromosome, the standard item part is just used to identify which items form which group.

The crossover in HGGA is as follows:

- Select two random crossing sites in each of the two parents.
- Inject the items of the crossing section of first parent at the first crossing site of the second parent.
- Eliminate any bins in the second parent which conflict with the injected bin.
- Use a local search mechanism to place eliminated bins back into the solution.

Falkenauer used a strategy similar to the domination criterion of Martello and Toth to place the eliminated bins (free items). Free items are swapped with non-free items (items currently placed within bins) such that the bins will consist of a few large items rather than many small items. Items that cannot be placed with this replacement strategy are re-inserted into the solution using a first-fit heuristic. Mutation works also similarly; it destroys a few bins from the population and reinserts the missing items using the mentioned local search procedure.

### 3. GENETIC ALGORITHMS

A Genetic Algorithm (GA) [5] is a search method to find approximate solutions to optimization problems. GAs use techniques inspired by the Darwinian theory of evolution such as crossover, mutation and natural selection. In a typical GA, a population of chromosomes (abstract representation of candidate solutions) goes through an evolutionary process. The most common representation scheme is binary encoding of strings of 0s and 1s. The evolution usually starts from a population randomly or heuristically initialized. In each generation, the quality of the solutions is evaluated by a fitness function; individuals are stochastically selected and are modified by crossover and mutation operators to form a new population. This process continues until the optimal solution is found or a termination criteria set by the user is met [44].

- 
- 1: Initialization of Chromosomes (individuals)
  - 2: Fitness Evaluation
  - 3: **repeat**
  - 4:   Selection of Chromosomes for Crossover
  - 5:   Crossover
  - 6:   Mutation
  - 7:   Fitness Evaluation
  - 8:   Population Update
  - 9: **until** Termination Criteria
- 

Figure 3.1. Pseudocode of a genetic algorithm

#### 3.1. Representation

There are many different ways to encode the chromosomes in the initial population. The most common approach is to encode the chromosome with fixed size binary strings as in Holland's original encoding method [5]. However, variable size and non-binary encodings are also present. The efficiency of the encoding method is problem dependent, thus should be adjusted according to the needs of the problem at hand.

a.	0	1	1	0	1	0	1	1	0
b.	0,1	2,5	5,5	3,2	-5,4	3,4	-4,1	7,9	0,8
c.	4	→ 8	→ 1	→ 9	→ 7	→ 2	→ 6	→ 5	→ 3

Figure 3.2. a) Binary representation b) Floating point representation c) Order based representation.

## 3.2. Selection

During at iteration, a number of chromosomes are selected to breed a new generation. Many selection methods are stochastic which ensure a small number of less fit individuals are selected as well. This helps to keep the diversity of the population. Some well known selection methods are as follows:

### 3.2.1. Roulette Wheel Selection

In the standard roulette wheel selection method [5], each chromosome has a chance of selection which is directly proportional to its fitness value. In a roulette wheel, each chromosome represents a pocket on the wheel and the size of each pocket is directly proportional to the probability of selection therefore selecting  $N$  chromosomes is like playing  $N$  games on the roulette wheel. The efficiency of this selection depends greatly on the range of the fitness values of the chromosomes. If the range is quite small then the likelihood of selection for each individuals will be very similar which may lead to stagnation in the search. This problem can be solved by using scaling techniques [45].

### 3.2.2. Rank Based Selection

In rank based selection [46], chromosomes in the population are sorted according to their fitness values. The selection probability of each individual is determined by the rank (its position on the sorted list) rather than the actual fitness value. The range of the ranks will be limited which prevents certain individuals to generate an excessive number of offspring. With a uniform scaling, ranking provides an effective control of selection pressure.

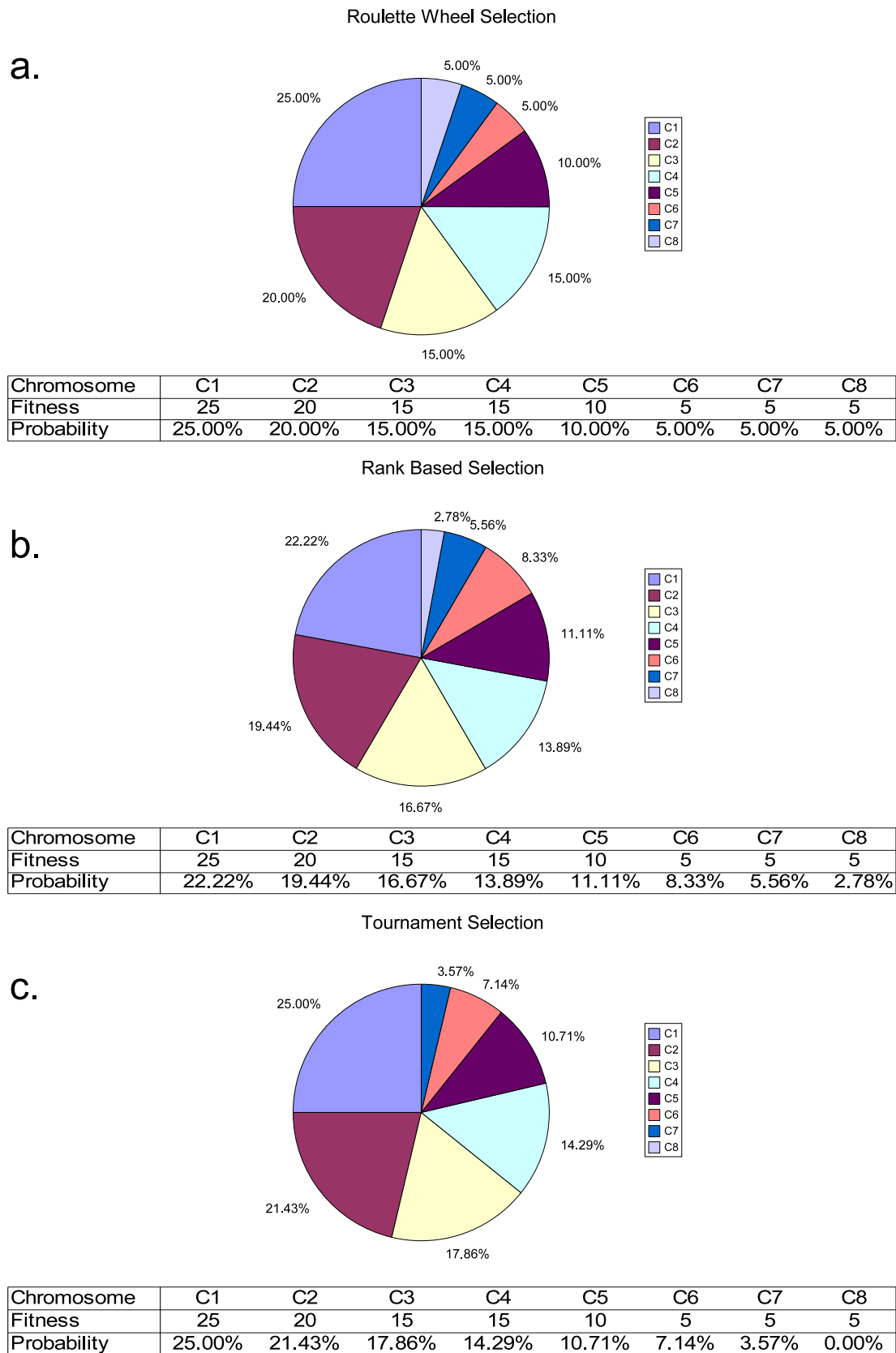


Figure 3.3. Selection probabilities for different selection methods a) Roulette wheel selection b) Ranking based selection c) Tournament selection with  $tour\ size = 2$ .

### 3.2.3. Tournament Based Selection

In tournament based selection [47],  $K$  individuals chosen randomly from the population are placed into a *tournament* and a winner is selected (usually the individual with the best fitness). Tournament selection has many advantages; it is easy to code and efficient since it does not require a sorting or scaling of individuals based on fitness as in ranking or roulette wheel selection. It works on parallel architectures and the selection pressure can easily be adjusted by decreasing or increasing the size of the tournament.

### 3.3. Crossover

Crossover operator is usually regarded as the most important operator in a genetic algorithm. Crossover is a recombination process in which exchange of segments of between two pairs of chromosomes called parents. Although there are many variants of crossover that are tailored to the needs of the problem at hand, here we present some of the most common crossover methods which can be regarded as *blind* (not problem specific).

- One Point Crossover: In one point crossover (1PTX) a crossover point is randomly selected on the parent chromosome. Then the values beyond that point in the chromosome are swapped between the two parent chromosomes.
- N Point Crossover: Unlike one point crossover,  $n$  crossover points are randomly selected on the parent chromosome. Then the values between odd and even crossover positions are swapped.
- Uniform Crossover: In uniform crossover each gene of the first parent has a fixed probability (usually 0.5) of swapping with the respective gene of the second parent.

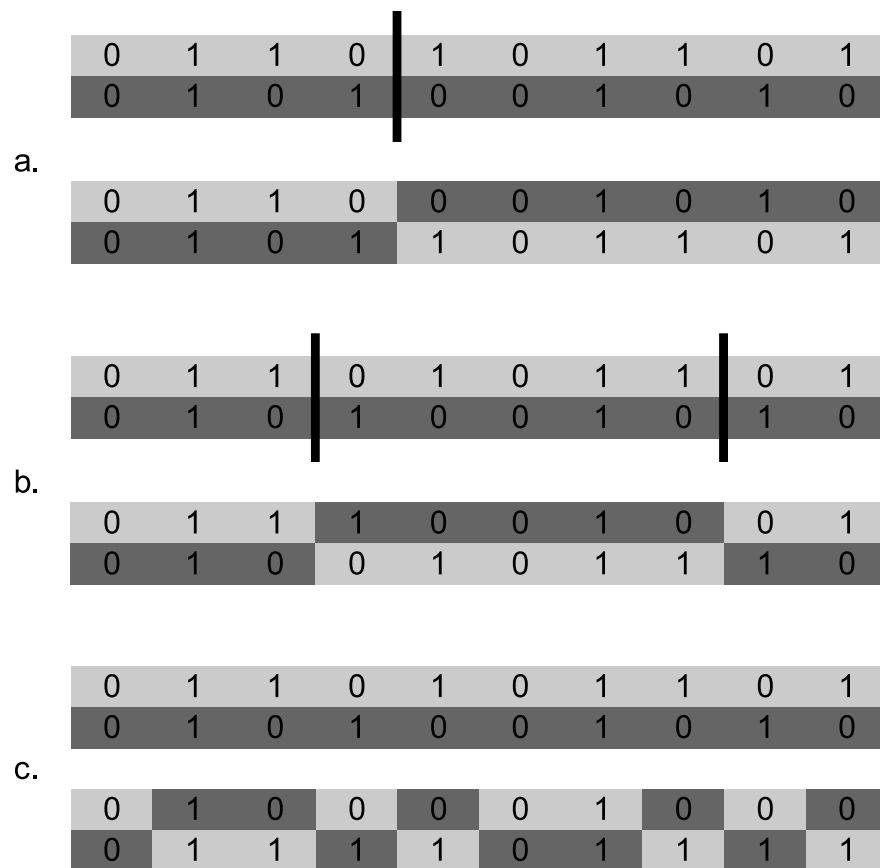


Figure 3.4. a) One point crossover b) N=2 point crossover c) uniform Crossover.

### 3.4. Mutation

Mutation is another important genetic operator whose purpose is to maintain the genetic diversity between successive generations of population. With just crossover and without mutation, it is usually impossible to generate new information not present in the population before unless crossover operator also has some mutational characteristics also. A local optima is avoided by using a mutation operator which prevents the over-similarity of the chromosomes in the population

The most traditional mutation in many binary coded representations is to simply flip the value of an arbitrarily chosen gene. In many implementations, mutation is given very low rates usually proportional to the length of the chromosome as too much mutation is



inherently destructive to the solution quality.

### 3.5. Replacement

After the offspring have been generated by genetic operators such as selection, crossover and mutation, those offspring have to be inserted into the population. Some common global replacement strategies are as follows [48]:

- Pure Reinsertion: Generate as many offspring as possible and replace all parents by this offspring.
- Uniform Reinsertion: Generate less offspring than parents and replace parents with a uniform distribution.
- Elitist Reinsertion: Generate less offspring than parents and replace the worst parents.
- Fitness Based Reinsertion: Generate more offspring than needed and reinsert the best offspring only.

Being the simplest replacement strategy, pure insertion is the simplest replacement mechanism, each individual lives only one generation. It is quite probable that good individuals will go extinct without producing any offspring. Combining elitism and fitness based reinsertion saves such individuals going extinct quickly. At each generation, a number of least fit individuals will be replaced with a number of best fit individuals. However, too much elitism may cause a diversity loss in the population, therefore a good number of new offspring has to be inserted to the later generations.

### 3.6. Memetic Algorithms

Similar to genetic algorithms, a memetic algorithm (MA) [49] is a population based meta-heuristic for solving optimization problems. Because MAs usually combine local search with crossover operators, they are also referred to hybrid genetic algorithms. Memetic algorithms can be categorized into two based on where the local search operators are applied. In a Baldwinian memetic algorithm [50], the local search is applied before the fitness is evaluated. The improvements of the local search are not saved in the individual and therefore

acquired traits of the parents are not inherited to the children. In a Lamarckian memetic algorithm [51], the local search is applied after the fitness is evaluated therefore acquired traits of the the parents influence the children. Lamarckian MA algorithms are usually faster in finding quality solutions with the risk of premature convergence. A Baldwinian MA would be more resistant to diversity loss in the population but it is usually much more slower than its Lamarckian counterpart.

It is assumed that a genetic algorithm is able to cover a broad range in the search landscape due to its population based nature, and a local search is able to find optimal solutions in promising parts of the landscapes. Due to a local search component, a memetic algorithm can also incorporate domain specific knowledge better than a blind genetic algorithm. Therefore memetic algorithms are usually much more efficient in terms of computing resources and tend to give state of the art results in many problems.

### 3.7. Multi-Objective Genetic Algorithms

Many real world optimization and search problems usually have multiple objectives. As a consequence, there is no unique best solution to such kind of a problem but a good-tradeoff between the solutions (the pareto optimal set). Evolutionary algorithms are desirable method for MOPs because of their population based structure which can represent multiple solutions to the problem.

A general multi-objective optimization problem can be defined as [52]:

$$\text{minimize objective function } F(X) = [f_1(x), f_2(x), \dots, f_k(x)] \quad (3.1)$$

with the  $m$  inequality constraints:

$$g_i(x) \geq 0 \quad i = 1, 2, \dots, m \quad (3.2)$$

and with the  $p$  equality constraints:

$$h_i(x) = 0 \quad i = 1, 2, \dots, p \quad (3.3)$$

where  $k$  is the number of objective functions  $f_i : R^n \rightarrow R$ . The vector  $x = [x_1, x_2, \dots, x_n]^T$  is called the decision variable vector.

Many multi-objective optimization algorithms use the concept of domination where two solutions are compared according to domination of one solution to another. Assume there are  $M$  objective functions in the problem [53]. Let the operation  $i \triangleleft j$  denote that  $i$  is better than a solution  $j$  on a particular objective function  $N$  in set  $M$ . Similarly let the operation  $i \triangleright j$  denote that the solution  $i$  is worse than  $j$  in that particular objective function  $N$  in set  $M$ . Also let operators  $\ntriangleleft$  and  $\ntriangleright$  denote the negative operations of  $\triangleleft$  and  $\triangleright$  respectively.

**Definition 1 (Pareto Dominance)** *A solution  $x_1$  is said to dominate another solution  $x_2$ , if following conditions are true:*

- *The solution  $x_1$  is no worse than  $x_2$  in all objectives: that is,*

$$f_j(x_1) \ntriangleright f_j(x_2) \quad \forall j = \{1, 2, \dots, M\} \quad (3.4)$$

- *The solution  $x_2$  is strictly better than  $x_1$  in at least one objective, or*

$$f_k(x_1) \triangleleft f_k(x_2) \quad \exists k = \{1, 2, \dots, M\} \quad (3.5)$$

If any one of the above conditions is violated, the solution  $x_1$  does not dominate the solution  $x_2$ .

The dominance relation is transitive but not reflexive, symmetric or antisymmetric [53]. Interestingly however, if a solution  $x_1$  does not dominate solution  $x_2$ , this does not imply that  $x_2$  dominates  $x_1$ .

**Definition 2 (Non Dominated Set)** *Among a set solutions  $P$ , the non-dominated set of solu-*

tions  $P'$  are those which are not dominated by any member of the set  $P$ .

If the set  $P$  is the entire search space or  $P = S$ , then the non-dominated set  $P'$  is called the *pareto – optimal* set. Like local and global optimal solutions in single-objective optimization problems, there are local and global pareto optimal sets in multi-objective optimization problems.

**Definition 3 (Globally Pareto-Optimal Set)** *The non dominated set of the entire feasible search space  $S$  is the globally pareto-optimal set.*

**Definition 4 (Locally Pareto-Optimal Set)** *If  $\forall x$  in a set  $P$ ,  $\nexists y$  dominating any member of the set  $P$  such that  $|y - x|_{\infty} \leq \epsilon$  where  $\epsilon$  is a small positive number, then solutions in set  $P$  form a locally pareto-optimal set.*

### 3.7.1. Implementations of Multi-objective Genetic Algorithms

In this section, some of the well-known multi-objective genetic algorithms are presented. A more detailed discussion of MOGAs can be found in [52].

**3.7.1.1. Non-Dominated Sorting Genetic Algorithm (NSGA).** In NSGA [54], the population is ranked according to non-domination. All non-dominated individuals (pareto front) are moved into the first category with a dummy fitness value proportional to the population size. To preserve the diversity of the population, the individuals are shared with their dummy fitness values. These first category individuals are later discarded and another category is formed from the non-dominated individuals remaining in the population. This process continues until all individuals are categorized. Since lower category individuals would have higher fitness values in this method, the likelihood of selection for crossover for these individuals are higher.

**3.7.1.2. Multi Objective Genetic Algorithm (MOGA).** In MOGA [55], the rank of an individual is determined by the number of individuals by which it is dominated. All non-dominated individuals are given rank 1. The fitnesses of the individuals are assigned by

interpolating from the best (rank 1) to the worst (rank  $n \leq M$ , where  $M$  is the population size).

3.7.1.3. Niche Pareto Genetic Algorithm (NPGA). In NPGA [56], two individuals are randomly selected from the population and compared against a small subset (comparison set) of individuals drawn from the population. If one individual is dominated by the comparison set and the other is not, then the latter is selected for crossover. If neither or both of the individuals are dominated by the comparison set, the winner is decided through fitness sharing.

---

```

1: candidate1 ← random Individual from Population
2: candidate2 ← random Individual from Population
3: cand1Dominated ← FALSE
4: cand2Dominated ← FALSE
5: for i ← 1 to comparisonSetSize do
6:   comparisonSet[i] ← random Individual from Population
7: end for
8: for i ← 1 to comparisonSetSize do
9:   if comparisonSet[i] dominates candidate1 then
10:    cand1Dominated ← TRUE
11:   end if
12:   if comparisonSet[i] dominates candidate2 then
13:    cand2Dominated ← TRUE
14:   end if
15: end for
16: if cand1Dominated = FALSE AND cand2Dominated = TRUE then
17:   Return candidate1
18: else if cand1Dominated = TRUE AND cand2Dominated = FALSE then
19:   Return candidate2
20: else
21:   nicheOfCandidate1 ← NicheCount(candidate1)
22:   nicheOfCandidate2 ← NicheCount(candidate2)
23:   if nicheOfCandidate1 < nicheOfCandidate2 then
24:     Return candidate1
25:   else
26:     Return candidate2
27:   end if
28: end if

```

---

Figure 3.5. Selection algorithm for niched pareto genetic algorithm

## 4. LINEAR LINKAGE ENCODING FOR GROUPING PROBLEMS

Many search and optimization problems exhibit symmetries in the search spaces. Such symmetries cause seemingly different solutions found by a search procedure to be equal to the same location in the search space. This is certainly undesirable since it would force a search algorithm to survey solutions which are already considered and thus would result an (exponential) increase in the time required for the algorithm. Therefore, many researchers have proposed symmetry breaking methods to prune redundant search spaces [57] [58].

In this chapter, a new representation scheme which eliminates the symmetry problem in grouping problems is explained. Linear Linkage Encoding (LLE) was first proposed for solving clustering problem by using a multi-objective genetic algorithm (MOGA) [7]. The researchers used LLE and MOGAs to minimize the number of the number of clusters and *total within cluster variation* (the sum of average distance of cluster elements to the center of the cluster). This new representation is able to represent solutions of various clusters in a fixed size chromosomes. Most importantly, LLE is able to reduce the search space considerably.

### 4.1. Previous Representations in Grouping Problems

Before moving onto the details of linear linkage encoding, previous work on the representational issues in grouping problems is considered. Most common representational schemes in grouping problems are as follows:

#### 4.1.1. Number Encoding

The most predominant representation in grouping problems is Number Encoding (NE) [59]. In NE, each object is encoded with a group id indicating which group it belongs to. For example the individual 2342123 encodes the solution where first object is in group 2, second in 3, third in 4, and so on. However, it is easy to see that the encoding 1231412 represents exactly the same solution, since the naming or the ordering of the partition sets is

irrelevant. The drawbacks of this representation are presented in [1] and it is pointed out that this encoding is against the minimal redundancy principles for the encoding scheme [60].

#### 4.1.2. Group Encoding

Another representation for grouping problems is Group Encoding (GE) [10]. The objects which are in the same group are placed into the same partition set. For instance, the sequence above can be represented as  $(1, 4, 6)(2, 7)(3)(5)$ . The ordering within each partition set is unimportant, since search operators work on groups rather than objects unlike in NE. However the ordering redundancy among groups still holds. For instance,  $(2, 7)(3)(5)(1, 4, 6)$  would again represent the same solution.

#### 4.1.3. Hybrid Grouping Genetic Algorithm representation

In the hybrid grouping genetic algorithm, Falkenauer [61] used a special representation for solving grouping problems. In HGGA standard number encoding is augmented with a group part encoding each group on a one gene for one group basis. For example the sequence above is represented as two parts: item part and group part  $1234567 : 1231412$  which means first item is in group 1, second item is in group 2. This has the same redundancy problem of group encoding as  $1234567 : 4124341$ . The difference from group encoding is that traditional search operators work on the group part of the encoding.

### 4.2. LLE Implementation

LLE can be implemented using an array. Let the entries in the chromosome be indexed with values from 1 to  $n$ . Each entry in the array then holds one integer value which is a link from one object to another object of the same partition set. With  $n$  objects, any partition set on them can be represented as an array of length  $n$ . Two objects are in the same partition set if either one can be reached from another through the links. If an entry is equal to its own index, then it is considered as an ending node. The links in LLE are unidirectional, thus; backward links are not allowed. In short, in order to be considered as a valid LLE array, the chromosome should follow the following two rules:



- The integer value in each entry is greater than or equal to its index but less than or equal to  $n$ .
- No two entries in the array can have the same value; the index of an ending node is the only exception to this rule.

Formally, let the set of elements to be clustered be  $E = \{e_1, e_2, \dots, e_n\}$  and let  $C = [g_1, g_2, \dots, g_n]$  be a sample chromosome in the population. Assume  $V$  is a function that denotes the value of a gene and  $I$  is the function which returns its index. Then, the following two properties hold for the LL encoding.

$$\forall g_i \in C [I(g_i) \leq V(g_i) \leq n]. \quad (4.1)$$

$$\forall g_i, g_j \in C [V(g_i) = V(g_j) \implies (i = j) \vee ((i > j) \wedge (V(g_i) = I(g_i)) \vee ((i < j) \wedge (V(g_j) = I(g_j)))] \quad (4.2)$$

If  $e_i$  and  $e_j$  are two objects where  $i < j$  then,

$$\varphi(e_i, e_j) = [V(g_i) = I(g_j)] \vee \exists g_k [(i < k < j) \wedge \varphi(e_i, e_j) \vee V(g_k) = I(g_j)] \quad (4.3)$$

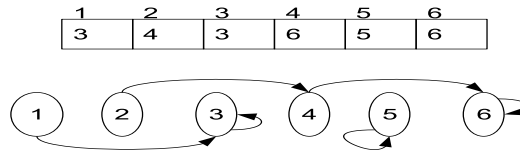


Figure 4.1. LLE array and LOP graphs

In LLE, the items in a group construct a linear path ending with a self referencing last item. It can be represented by the *labeled oriented pseudo (LOP) graph*. A LOP Graph is a labeled directed graph  $G(V, E)$ , where  $V$  is the vertex set and  $E$  is the edge set. A composition of  $G$  is a grouping of  $V(G)$  into disjointed oriented pseudo path graphs  $G_1, G_2, \dots, G_m$  with the following properties:

- Disjoint paths:  $\bigcup_{i=1}^m V(G_i) = V(G)$  and for  $i \neq j$ ,  $V(G_i) \cap V(G_j) = \emptyset$
- Non-backward oriented edges: If there is an edge  $e$  directed from vertex  $v_i$  to  $v_k$  then  $i \leq k$ .
- Balanced Connectivity
  - a.  $|E(G)| = |V(G)|$
  - b. each  $G_i$  has only one ending node with an *in-degree* of 2 and *out-degree* of 1.
  - c. each  $G_i$  has only one starting node whose *in-degree* = 0 and *out degree* = 1
- All other  $|V(G_i)| - 2$  vertices in  $G_i$  have *in-degree* = *out-degree* = 1.

Du [62] and Korkmaz [63] have used the following to prove that there is a unique mapping between possible partitions and the chromosomes of LLE.

**Theorem 1** *Given as set of items  $S$ , there is a one to one mapping between the chromosomes of LLE and the possible partitions.*

**Lemma 1** *Linear linkage encoding is an implementation of the LOP graph.*

**Proof 1** *Each gene in the LLE has one out-link therefore an item represented by a gene can belong to only one group. LLE represents disjoint partition sets. Hence, the first property of LOP is satisfied. The second property of the LOP graph is equivalent to the first constraint of LLE. The balanced connectivity property of the LOP graph is due to the second constraint of LLE.*

**Lemma 2** *Given a set of items  $S$ , there is one and only one composition of LOP Graphs  $G(V, E)$  for each grouping of  $S$ , where  $|V| = |S|$ .*

**Proof 2** *Let the items in  $S$  be indexed from 1 to  $n$  and consider a partition  $P(S)$  of  $S$ . Assuming the items in each group are ordered in ascending order and each item is directed to the next higher indexed item in that group. Adding a self-referencing link to the last item in that cluster, we will have a LOP graph. Therefore, there exists at least one composition of  $G$  for each partition. Since all index values are distinct, there can be only one possible ascending order within a group for a possible partition. Hence, there exists only one and one only composition of  $G$  for each partition. By definition, a LOP graph represents only a single partition.*

Based on Lemma 1 and Lemma 2, in LLE there is a one-to-one mapping between the chromosomes and partitioning solutions.

**Corollary 1** *The number of chromosomes corresponding to all possible partitions is given by the  $n^{\text{th}}$  Bell number.*

The number of ways that a set of  $n$  elements can be partitioned into disjoint nonempty subsets is called the Bell number which is denoted by  $B(n)$ . We have therefore

$$B(n) = \sum_{k=0}^n S(n, k) \quad n \geq 1 \quad (4.4)$$

where  $S(n, k)$  are the Stirling numbers of the second kind. Stirling numbers is  $S(n, k)$  is the number of ways that a set of cardinality  $n$  can be partitioned into exactly  $k$  nonempty subsets. Stirling numbers of the second kind can be calculated as follows:

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^i \binom{k}{i} (k-i)^n \quad (4.5)$$

According to the first observation, there is a one to one correspondence between the chromosomes in LLE and the possible partition sets. The number of different chromosomes in LLE then can be denoted by the  $n^{\text{th}}$  Bell number  $B(n)$ . Compared to number encoding, LLE allows a  $\frac{n^n}{B(n)}$  times smaller solution space.

### 4.3. Repair Procedure for Linear Linkage Encoding

Traditional operators in many search methods may violate the two key properties of Linear Linkage Encoding mentioned before. To repair backward links, the procedure moves from the first element to the last. If it sees a backward link, then that link is reversed. To repair the entries pointing to the same value, the repair procedure moves from the last item of the array to the beginning. At each step, algorithm checks whether there is any element same with the current value from that position to the end of the array. If so, it reassigns the value of the element to the point where previous value is same.

---

**Require:** One chromosome *nonLLEchromosome* violating LLE rules.

**Ensure:** One chromosome *LLEchromosome* conforming to LLE rules.

```

for  $i \leftarrow 1$  to arraysize do
  if  $value[i] < i$  then
     $value[value[i]] \leftarrow i$ 
  end if
end for
for  $i \leftarrow arraysize$  to 1 do
   $repairMode \leftarrow TRUE$ 
   $k \leftarrow value[i]$ 
  if  $k \neq i$  then
    repeat
      for  $j \leftarrow i + 1$  to  $j < k$  do
        if  $value[i] \neq value[j]$  then
           $value[i] \leftarrow k \leftarrow j$ 
           $repairMode \leftarrow TRUE$ 
          Exit for
        end if
      end for
    until  $repairMode = false$ 
  end if
end for

```

---

Figure 4.2. Repair procedure for linear linkage encoding

#### 4.4. Initialization

In the original LLE [7], authors used a random initialization method where each gene in a chromosome is assigned a value between 1 and the number of objects  $n$ . However such a method produces chromosomes which violate the rules of LLE. Therefore a repair procedure described above has been used to correct such violations. However such an initialization

tend to result in chromosomes with few number of groups. In order to rectify it, authors have manually injected a number of chromosomes with higher number of groups into the population.

#### 4.5. Crossover and Mutation

In the first implementation of LLE [7], researchers used traditional one point crossover which allows different clusters to exchange partial contents. One point crossover can also split a cluster into two.

As a mutation operator, researchers used a new mutation method which they called as grafting mutation. In grafting mutation, the membership of set of objects rather than just a single object is changed. The details of this mutation is described in Figure 4.3.

---



---

```

1:  $g \leftarrow \text{Random}(1, N)$ 
2:  $C_i \leftarrow \text{GroupOf}(g)$ 
3:  $e \leftarrow \text{EndingNodeOf}(C_i)$ 
4:  $m \leftarrow \text{Random}(1, \text{numberOfClusters})$ 
5: if  $m = \text{numberOfClusters}$  AND  $g \neq e$  then
6:   Split  $C_i$  into  $C_{i1}$  and  $C_{i2}$ 
7:    $\text{EndingNodeOf}(C_{i1}) \leftarrow g$ 
8: else
9:   Split  $C_i$  into  $C_{i1}$  and  $C_{i2}$ 
10:   $\text{EndingNodeOf}(C_{i1}) \leftarrow g$ 
11:   $\text{EndingNodeOf}(C_{i2}) \leftarrow e$ 
12:   $f \leftarrow \text{Random}(e, g)$ 
13:   $\text{value}[f] \leftarrow \text{EndingNodeOf}(m)$ 
14: end if

```

---

Figure 4.3. Grafting mutation in linear linkage encoding

## 5. A MULTI-OBJECTIVE GENETIC ALGORITHM FOR GRAPH COLORING AND TIMETABLING

The main intention in this chapter is to propose a multi objective solution foundation to multi-constraint timetabling problems. None of the efficient graph coloring algorithms in the literature empowers genetic operators as their main search mechanism. These methods usually rely on local search operators. Therefore this chapter deals with the applicability of linear linkage encoding on grouping problems by using suitable crossover and mutation operators. A multi-objective genetic algorithm employing a weak elitism is used and the main search operator of this approach is a mutation aided by a crossover operator [64].

### 5.1. Initialization

Since the proposed method is tested on a minimal coloring problem, it is desirable to initialize the population with individuals having different number of colors. Setting the range of number of colors too wide will unnecessarily increase the search space and thus the execution time. For example, in a graph instance with 128 vertices and with a predicted chromatic number of 20, setting the range between the minimum and maximum values ( $l$  and the  $|V|$  which is 128 in this case) will force the algorithm to survey individuals far away from the optimal coloring. It is also undesirable to set the range too narrow either. Such a scheme will prevent promising individuals with different number of colors from cooperating through crossover and mutation.

There are various methods for finding tight lower and upper bounds by using cliques, maximal degree of the graph, and so on [65]. Since exact or approximate chromatic numbers in the test instances are already known, these bounds are set manually in this study.

In the experiments, a population with individuals having different number of colors and an external population which holds the best individuals with the minimal conflicts for a specific number of colors within a search range ( $lowerBound \leq k \leq upperBound$ ) are

used. In order to create an individual, first  $k$  is determined, then a  $k$ -colored individual is randomly created. An external smart initialization method is not used to reduce the edge conflicts in order not to give any bias to our crossover operators and to let the multi-objective evolutionary method do the search.

## 5.2. Selection

A  $k$ -coloring problem is solved when the number of conflicting edges is zero. If a  $k$ -coloring solution is obtained,  $k + 1$  colorings can also be generated by dividing independent sets into two. It might be possible to unite two sets in a  $k+1$  coloring to obtain a  $k$ -coloring. Therefore, if the chromatic number lies within the upper and lower bounds, the pareto front will almost be a straight line along the color axis with zero conflict if the lower bound is set close to the chromatic number. A restricted multi-objective method might work efficiently on a search range within specified bounds around the chromatic number.

As a multi-objective genetic algorithm, a modified version of Niche Pareto Genetic Algorithm (NPGA) described in [56] is used. In NPGA, two candidate individuals are selected at random from the population to be one of the mates. A comparison set is formed from randomly selected individuals within the population. Each candidate is then compared against each individual in the comparison set. If one candidate is dominated by the comparison set (which means it is worse for every part of the objective function than any individual in the comparison set) and the other is not, then the latter is selected for reproduction. If neither or both are dominated by the comparison set, then niching is used to select a winner mate. The size of comparison set ( $t_{dom}$ ) allows a control over the selection pressure. The comparison set size is preset to around ten percent of the population size as suggested in [56].

When neither or both candidates are dominated by the comparison set, the candidate with a smaller niche count is selected for reproduction. The niche value  $m_i$  of the  $i^{th}$  indi-

vidual is calculated by:

$$m_i = \sum_{j \in pop} sh(d[i, j]) \quad (5.1)$$

where  $d[i, j]$  is the distance between two individuals according to objective function values and  $sh(d)$  is the sharing function which is:

$$sh(d) = \begin{cases} 1 & \text{if } d = 0 \\ 1 - d/\mu_{share} & \text{if } d < \mu_{share} \\ 0 & \text{if } d \geq \mu_{share}. \end{cases} \quad (5.2)$$

and distance measure is Manhattan distance in terms of color and conflict values in the individuals.

$$d[i, j] = |c_{i1} - c_{j1}| + |c_{i2} - c_{j2}| \quad (5.3)$$

### 5.3. Objective Functions

The objective functions used in this study are quite straightforward. One of them measures the number of color sets and the other measures the number of conflicting vertices. The goal is to minimize both of this contradictory objective functions.

### 5.4. Redundancy and Genetic Operators

Although LLE is a non-redundant representation for grouping problems in theory, this advantage practically disappears if the search operators do not adhere to this principle. Therefore, a more desirable option is to make the search non-redundant additional to the representation. For example, consider a basic hill climbing mutation which sends one vertex from one set to another. This is analogous of changing a gene value in the number encoding. If majority of the group ids of the items can be maintained for a long period of time, then



it is quite possible to make a low-redundant search even on a highly redundant encoding such as NE. This is one of the reasons local search based methods are quite successful on grouping problems. Because of the small perturbations on the search space, these methods not only preserve the building blocks on the candidate solution but also are able to operate on a low-redundant small region of the large search landscape.

The same advantage diminishes for a crossover operator which causes huge jumps on the search space. It is possible to keep the majority of the group ids of the items fixed by using traditional crossovers like one-point or uniform crossover. Such methods, however do not preserve the groups which are the building blocks themselves. A crossover operator should preserve the order of the colors as long as possible. Two ordering mechanism which assigns group ids to the groups after crossover and mutation are investigated within the context of LLE. These two redundancy elimination mechanisms are based on the cardinality of the groups and the lowest index number at each group. In [11], the authors investigated the effect of these two methods on Graph Coloring by using 0/1 integer linear programming SAT solvers.

#### **5.4.1. Cardinality Based Ordering**

In Cardinality Based Ordering, each group receives a group id according to its cardinality (set size). Groups are sorted according to their cardinality and the group with the highest cardinality will be assigned group id 1, the second highest will be identified as group 2, and so on. For example groups  $(1, 3)(5)(2, 4, 6)$  are indexed as  $V_1 = (2, 4, 6)$ ,  $V_2 = (1, 3)$ , and  $V_3 = (5)$ . Since more than one group can have the same cardinality, the ordering might not be unique.

#### **5.4.2. Lowest Index Ordering**

In Lowest Index Ordering, the smallest index in each group is found first, then the group with the smallest index number is assigned group id 1, the group with the second smallest index number is assigned group id 2, and so on. For example, groups  $(1, 3)(5)(2, 4, 6)$  are indexed as  $V_1 = (1, 3)$ ,  $V_2 = (2, 4, 6)$ , and  $V_3 = (5)$ . Since each group has one unique

lowest index, the ordering is always unique.

## **5.5. Crossover**

Linear linkage encoding can be implemented using one dimensional arrays, allowing applicability of the traditional crossover methods such as, one point or uniform crossover. However, it is observed during experimentations that these crossovers can be too destructive especially for graph coloring due to the danger of introducing new links in the LOP graph absent in both parents. Also since the building blocks [45] in graph coloring are strictly large independent sets (not even independent set segments), there is a risk of destructing these building blocks. However, for small problem instances, one-point crossover in LLE is reported to generate satisfactory results for clustering problem [7]. (This might be due to the fact that building blocks may be a segment of clusters rather than the whole cluster.) Three types of crossover operators are compared using LLE representation.

### **5.5.1. Greedy Partition Crossover**

Graph Coloring Problem can be considered as partitioning the graph into independent sets. Therefore, by preserving the large independent sets, the vertices in non-independent sets can be forced to form independent sets as well.

Greedy Partition Crossover (GPX) was proposed by Galinier and Hao [10] in their Hybrid Graph Coloring Algorithm. The idea is to transmit the largest set (group) from one parent, then to delete the vertices in this largest set from the other parent. This transmission and deletion process is repeated on both parents successively until all of the vertices are assigned to the child.

Two forms of Greedy Partition Crossover by following the rules of Cardinality and Lowest Index Ordering are implemented. The difference is just assigning the color ids to the groups after the crossover. In GPX Lowest Index Crossover (GPX-LI), the groups with lower index numbers are given lower color ids, whereas in GPX Cardinality Based Crossover (GPX-CB), the lower color ids are assigned to the groups with higher cardinality. A general

pseudocode of GPX is presented in Figure 5.1.

Consider two parents in Figure 5.2. One can obtain the child as follows: Largest Set in parent 1 is (3, 4, 5, 6). This set is transmitted to the child and 3, 4, 5 and 6 are deleted from parent 2. After this deletion largest set in parent 2 (1) is transmitted to the child. Finally (2) is assigned as the last group. After sorting according to lowest index ordering (GPX-LI), the coloring then becomes  $C_1 = (1)$ ,  $C_2 = (2)$ ,  $C_3 = (3, 4, 5, 6)$ . If the groups are sorted according to their cardinality (GPX-CB), the coloring is  $C_1 = (3, 4, 5, 6)$ ,  $C_2 = (1)$ ,  $C_3 = (2)$ . The time complexity of GPX is  $O(k^2 + n)$  where  $k$  is the number of groups and  $n$  is the number of vertices.

---

**Require:** Two Parents -  $parent1$  and  $parent2$  in LLE form.

**Ensure:** One *offspring* in LLE form.

```

1:  $currentParent \leftarrow \text{Random}(parent1, parent2)$ .
2: repeat
3:    $largestSet \leftarrow \text{Find largest set in } currentParent$ .
4:   transmit unassigned the vertices (links) in the  $largestSet$  to offspring.
5:   mark transmitted vertices as assigned.
6:   if  $currentParent = parent1$  then
7:      $currentParent \leftarrow parent2$ .
8:   else
9:      $currentParent \leftarrow parent1$ .
10:  end if
11: until all vertices are assigned
12: if Lowest Index Ordering is Used then
13:   sort group ids according to lowest index number (GPX-LI).
14: else
15:   sort group ids according to cardinality (GPX-CB).
16: end if

```

---

Figure 5.1. Pseudocode of the greedy partition crossover

Both GPX-LI and GPX-CB are applicable to other representations such as number or group encodings. Our intention of using these crossovers is to create crossover operators applicable only to LLE. The following two crossovers are inspired from GPX.

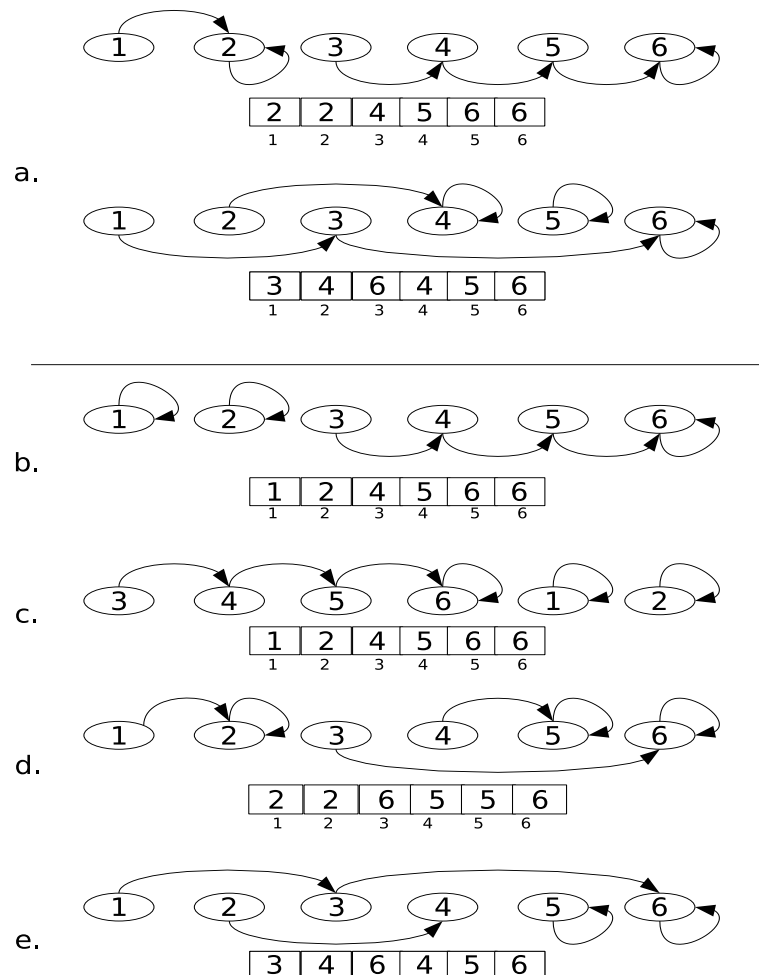


Figure 5.2. a) Two Parents in LLE array and LOP graph form. b) Resulting offspring from greedy partition crossover - lowest index ordering c) Resulting offspring from greedy partition crossover - cardinality based ordering. d) Resulting offspring from lowest index first crossover. e) Resulting offspring from lowest index max crossover.

### 5.5.2. Lowest Index First Crossover

In Lowest Index First Crossover (LIFX), the goal is to transmit the groups beginning with lowest index numbers. LIFX works as follows:

A parent is randomly selected. Beginning with the lowest index (vertex) which has not been assigned yet, the vertices are transmitted to the child by following the links. If the vertices along the path are assigned before, they are skipped. The process is repeated by successively changing the parents for transmission until all of the vertices are assigned to the child. A general pseudocode of LIFX is presented in Figure 5.3.

---

**Require:** Two Parents -  $parent1$  and  $parent2$  in LLE form.

**Ensure:** One  $offspring$  in LLE form.

```

1:  $i \leftarrow 1$ 
2: repeat
3:    $currentParent \leftarrow \text{Random}(parent1, parent2)$ .
4:    $lengthOfParent \leftarrow$  Calculate the path length of  $currentParent$  starting from  $i$ .
5:   transmit unassigned vertices (links) in the  $parentToSelect$  to  $offspring$ .
6:   mark transmitted vertices as assigned.
7:    $i \leftarrow$  next unassigned vertex.
8:   if  $currentParent = parent1$  then
9:      $currentParent \leftarrow parent2$ .
10:  else
11:     $currentParent \leftarrow parent1$ .
12:  end if
13: until all vertices are assigned

```

---

Figure 5.3. Pseudocode of the lowest index first crossover

The application of LIFX on the parents in Figure 5.2 would be as follows: Assuming we begin with first parent, current lowest index number is 1. Therefore, (1, 2) is transmitted to the child. The current lowest index number is now 3. Switching to parent 2, we copy (3, 6) as the next group. Switching back to parent 1, current lowest index is 4, therefore (4, 5) is copied to the child. Final coloring then becomes:  $C_1 = (1, 2)$ ,  $C_2 = (3, 6)$ ,  $C_3 = (4, 5)$ . The time complexity of LIFX is  $O(n)$  where  $n$  is the number of vertices.

Note that this crossover prioritizes groups beginning with the lowest index number,

therefore it reduces the sizes of the groups beginning with higher index numbers. This is in concordance with the nature of LLE, because the number of possible values for the higher index locations is lower.

### 5.5.3. Lowest Index Max Crossover

In Lowest Index Max Crossover (LIMX), the child is generated with two objectives: Transmit large groups to preserve Cardinality Based Ordering, and to transmit groups beginning with lowest index number (to preserve Lowest Index Ordering). Therefore this method can be considered as an amalgamate of LIFX and GPX. LIMX works as follows:

---

**Require:** Two Parents -  $parent1$  and  $parent2$  in LLE form.

**Ensure:** One *offspring* in LLE form.

```

1:  $i \leftarrow 1$ 
2: repeat
3:    $lengthOfParent1 \leftarrow$  Calculate the path length of  $parent1$  starting from  $i$ .
4:    $lengthOfParent2 \leftarrow$  Calculate the path length of  $Parent1$  starting from  $i$ .
5:   if  $LengthOfParent1 < LengthOfParent2$  then
6:      $parentToSelect \leftarrow parent1$ .
7:   else
8:      $parentToSelect \leftarrow parent2$ .
9:   end if
10:  transmit unassigned vertices (links) in the  $parentToSelect$  to offspring.
11:  mark transmitted vertices as assigned.
12:   $i \leftarrow$  next unassigned vertex.
13: until all vertices are assigned

```

---

Figure 5.4. Pseudocode of the lowest index max crossover

Beginning with the lowest index number (vertex) which has not been assigned first we calculate the length of the links (path length) in both parents. Already assigned vertices

are not counted in this link length calculation. This allows finding the largest set in parents beginning with the lowest index number. Then the links (and thus vertices) are transmitted to the child from the parent with the greater link-length. After that next unassigned lowest index number is found and the process is repeated until all vertices are assigned. A general pseudocode of LIMX is presented in Figure 5.4.

Application of LIMX to parents in Figure 5.2 is as follows: Current lowest index is 1. (1, 3, 6) is longer than (1, 2) so (1, 3, 6) is copied to the child. Current lowest index is now 2. (2, 4) is larger than (2) so it is transmitted to the child. Finally (5) is copied to the child as the last group. At the end of LIMX the coloring then becomes:  $C_1 = (1, 3, 6)$ ,  $C_2 = (2, 4)$ ,  $C_3 = (5)$ . The time complexity of LIMX is  $O(n)$  where  $n$  is the number of vertices.

## 5.6. Mutation

In this study, a mutation scheme that sends a selected conflicting vertex  $x$  from its color set to the best possible other one is used. A tournament method is used to select a vertex for transfer. A percentage of conflicting vertices are taken into a tournament and the vertex with the highest conflict in this set is transferred to a best color available.

This mutation can increase or decrease the number of colors by one. The number of colors will increase if a one vertex group is introduced by the operator and it will decrease if a one vertex group is merged with another one.

As aforementioned, assigning group ids after crossover is essential for low redundancy and the success of the mutation. In GPX-LI, LIMX and LIFX, the ids are assigned according to Lowest Index Ordering whereas in GPX-CB the ids are assigned according to Cardinality Based Ordering.

## 5.7. Replacement

In the simulations, a trans-generational replacement with weak elitism is employed. At each generation,  $\lambda$  offspring are produced by  $\lambda$  (non elitist) +  $\mu$  (elitist individuals, one

for each number of colors within the searching range) parents to replace the  $\lambda$  non elitist parents. By using elitism to hold the best conflict per number of colors a bias to number of colors constraint is given.

## 5.8. Experimental Results

Several graphs from the DIMACS Challenge Suite [9] are used in these tests. The general test setup is summarized in Table 5.1.

Table 5.1. Test setup for graph coloring instances

<b>Test Machine:</b>	Pentium 4 2Ghz with 256MB Ram
<b>Compiler:</b>	GCC C++ 3.2 with -O2 flags
<b>No of Generations:</b>	10000
<b>Population Size:</b>	25 percent of the number of vertices in graph
<b>Comparison Set Size:</b>	10 percent of the population size
<b>Niche Size:</b>	5.0
<b>Crossover Rate:</b>	0.25
<b>Mutation Rate:</b>	a single mutation is enforced
<b>Number of Runs:</b>	50 for each instance

In Table 5.2, the characteristics of the test instances sampled from the DIMACS test suite are presented. Table shows the name, number of vertices ( $|V|$ ), number of edges ( $|E|$ ), edge density (%) and chromatic number ( $\chi(G)$ ) of the instances.

In all the tests, the mutation count is set to 1, and crossover rate is fixed at 0.25. Higher crossover rates have negative impact in the performance. In this setup, the algorithm is more like a genetic hill climbing method. Since the chromatic number of these graphs are already known, the range is set manually according to the chromatic number  $\chi(G)$ .

Note that the primary intention is to compare the crossover operators in the context of



Table 5.2. Data characteristics of the problem instances from the DIMACS suite

Instance	$ V $	$ E $	%	$\chi(G)$
DSJC125.5	125	3891	0,50	?
DSJC125.9	125	6961	0,90	?
zeroin.1.col	211	4100	0,19	49
zeroin.2.col	211	3541	0,16	30
zeroin.3.col	206	3540	0,17	30
DSJC250.1	250	3218	0,10	?
DSJC250.5	250	15668	0,50	?
DSJC250.9	250	27897	0,90	?
flat300_20	300	21375	0,48	20
flat300_26	300	21633	0,48	26
flat300_28	300	21695	0,48	28
school1_nsh	352	14612	0,24	14
le450_15a	450	8168	0,08	15
le450_15b	450	8169	0,08	15
le450_15c	450	16680	0,17	15
le450_15d	450	16750	0,17	15
le450_25a	450	8260	0,08	25
le450_25b	450	8263	0,08	25
le450_25c	450	16680	0,17	25
le450_25d	450	16750	0,17	25
DSJC500.1	500	12458	0,10	?
DSJC500.5	500	62624	0,50	?

LLE. As a result, the experiments are run for a limited period of time. (The longest time required for one run is around 5 minutes for cars91 problem instance). This might have resulted in performance hit for large problem instances which may need an increase in the maximum number of generation.

Unfortunately, a very poor performance from one point crossover has been observed in the experiments. It was not even able to generate solutions in the color search range specified. Therefore the experimental results of one point crossover are not included.

In Table 5.3, the best solutions obtained after 50 runs in DIMACS instances by using the four crossover operators are presented. Figure 5.5 represents the average color number of 50 runs for each instance in the DIMACS test suite. The results show no significant statistical differences between crossover operators except for a few instances. For example for flat300\_20 graph, LIMX was able to find a best 20 coloring while the other crossovers were very far from the optimal. However, for this graph, average colorings found with all crossovers and standard deviation are quite high. This is possibly due to the natural difficulty of flat graphs. Another slight difference appeared in register allocation graphs (zeroin.X.col graphs) where LIFX performed worst while GPX crossovers performed best.

The performance of different crossover operators that can be used with LLE is analyzed. It is also important to compare these with the current results in the literature. Graph coloring algorithm results of Kirovski et al. [66] for two set of parameters (Kirovski B and Kirovski C) are given in Table 5.3 for this comparison. Kirovski's algorithm is based on divide and conquer paradigms, global search for constrained independent sets, assignment of most-constrained vertices to least constraining colors, reuse and locality exploration of intermediate solutions, post processing lottery-scheduling iterative improvement. Another important aspect is that Kirovski's algorithm is not dependent on a fixed  $k$  for the number of colors. With respect to Kirovski's solutions, the crossovers gave similar and for some instances better results however when the instance becomes larger and more difficult, Kirovski's algorithm performs better.

Table 5.4 presents some instances taken from the Carter's Benchmark [8]. We again

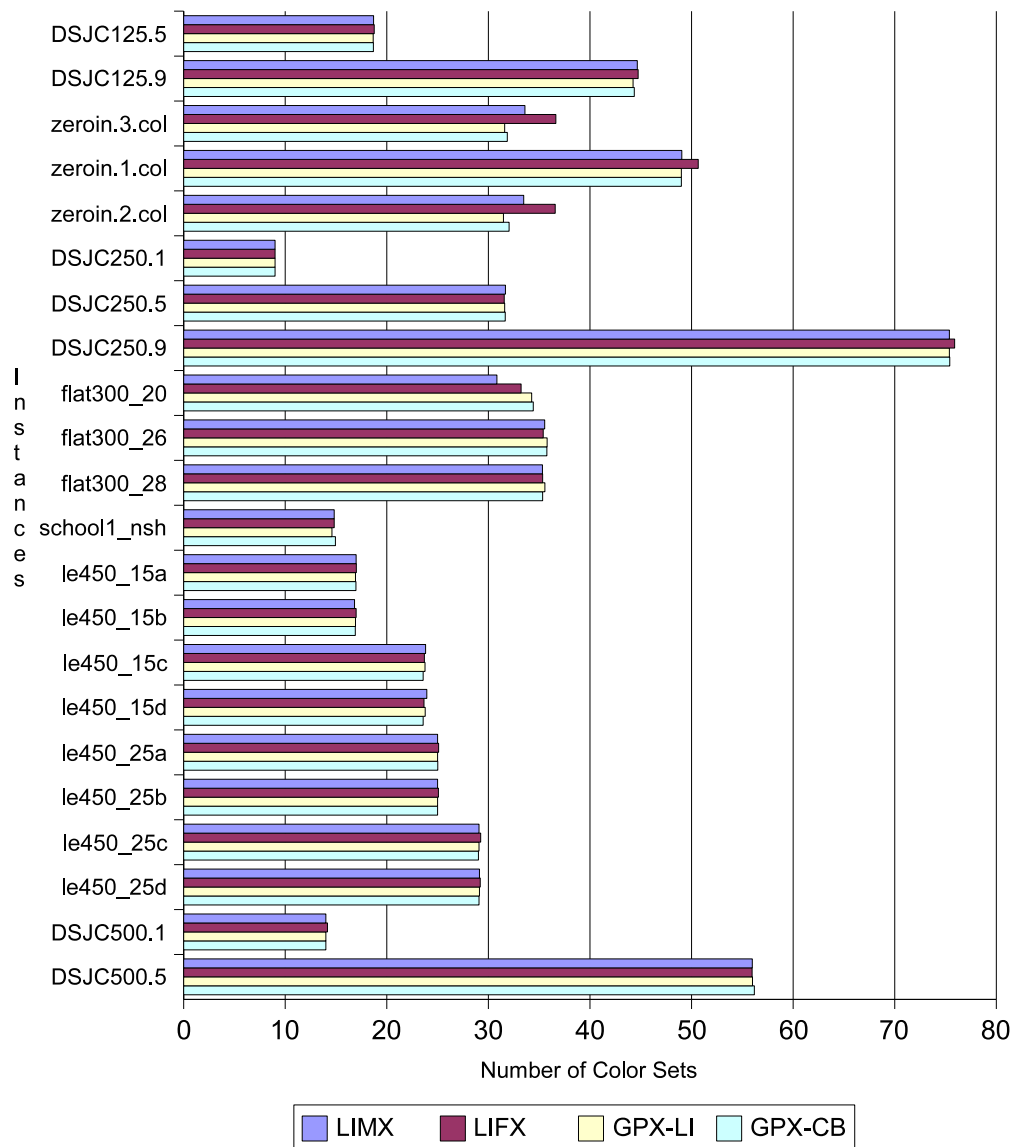


Figure 5.5. Average number of groups for instances in DIMACS benchmark.

Table 5.3. Best colorings obtained for the instances in the DIMACS benchmark suite

Instance	$\chi(G)$	LIMX	LIFX	GPX-LI	GPX-CB	Kirovski-B	Kirovski-C
DSJC125.5	?	18	18	18	18	19	18
DSJC125.9	?	44	44	44	44	45	45
zeroin.1.col	49	49	50	49	49	49	49
zeroin.2.col	30	31	35	31	31	30	30
zeroin.3.col	30	31	35	30	31	30	30
DSJC250.1	?	9	9	9	9	9	9
DSJC250.5	?	31	31	31	31	30	30
DSJC250.9	?	75	75	75	74	77	77
flat300_20	20	20	31	27	32	20	20
flat300_26	26	34	34	34	34	32	28
flat300_28	28	34	34	34	34	33	32
school1_nsh	14	14	14	14	14	16	14
le450_15a	15	16	16	16	16	17	17
le450_15b	15	16	16	16	16	17	17
le450_15c	15	23	23	23	23	22	21
le450_15d	15	23	23	23	23	22	21
le450_25a	25	25	25	25	25	25	25
le450_25b	25	25	25	25	25	25	25
le450_25c	25	28	29	28	28	28	28
le450_25d	25	28	28	28	28	?	?
DSJC500.1	?	14	14	14	14	14	14
DSJC500.5	?	55	55	55	55	51	50

Table 5.4. Data characteristics of the problem instances from the Carter benchmark suite

Instance	$ V $	$ E $	%
Hecs92	81	1363	0.42
Staf83	139	1381	0.14
Yorf83	181	4691	0.29
Utes92	184	1430	0.08
Earf83	190	4793	0.27
Tres92	261	6131	0.18
Lsef91	381	4531	0.06
Kfus93	461	5893	0.06
Ryes93	486	8872	0.08
Carf92	543	20305	0.14
Utas92	622	24249	0.13
Cars91	682	29814	0.13

present the number of vertices, edges and edge density of these graphs in this table. Table 5.5 represents the best colorings obtained after 50 runs. In Figure 5.6, the average colorings of 50 runs are presented.

For the problem instances in the Carter's timetabling benchmark, again, a significant difference among crossover operators is not observed. However, LIMX has a slightly better performance in terms of best and average color (group) number. LIMX gave the best colorings in staf83 and lsef91 instances while others were one color behind it. Yet, the difference between average colorings and standard deviation is not statistically significant for almost all instances.

The best colorings after 10000 generations are compared with some of the results from the literature (Carter et al. [8], Caramia et al. [67] and Merlot et al. [68]) in Table 5.5. Like DIMACS instances, the performance of the graphs with vertices above 500 suffered due to the limit on the maximum number of generations. For problem instances, the crossovers

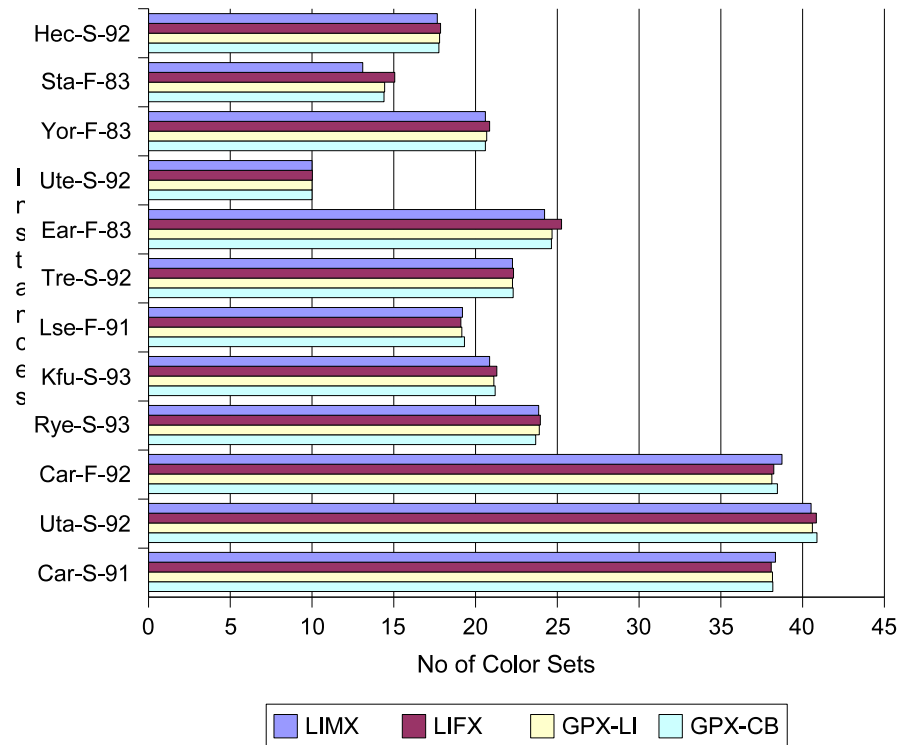


Figure 5.6. Average number of groups obtained for instances in Carter's Benchmark.

Table 5.5. Best colorings obtained for the instances in the Carter's benchmark suite

Instance	LIMX	LIFX	GPX-LI	GPX-CB	Carter	Caramia	Merlot
Hecs92	17	17	17	17	17	17	18
Staf83	13	14	14	14	13	13	13
Yorf83	20	20	20	20	19	19	23
Utes92	10	10	10	10	10	10	11
Earf83	23	24	24	23	22	22	24
Tres92	21	21	21	21	20	20	21
Lsef91	17	18	18	18	17	17	18
Kfus93	20	20	20	20	19	19	21
Ryes93	23	23	23	23	21	21	22
Carf92	36	36	36	36	28	28	31
Utas92	38	39	38	38	32	30	32
Cars91	36	36	37	35	28	28	30

gave similar results in terms of best grouping obtained. Generally they obtained colorings equal or one color behind colorings of Carter et. al and Caramia et. al, and better than of Merlot et. al.

Full test results are presented with average best coloring found, standard deviation, and best coloring of 50 runs for each crossover operator in Appendix A.

## 6. A HYBRID GENETIC ALGORITHM FOR BIN PACKING

In this chapter we present a single-objective genetic algorithm for bin packing problem. The algorithm incorporate LLE as a representation, one point crossover for a recombination operator and utilizes first-first decreasing heuristic with an additional replacement mechanism for well-filled bins.

### 6.1. Initialization

The individuals are initialized using a first fit heuristic described in Chapter 2 and resulting chromosome is converted to LLE form. In order not to have the same individual for the whole population, a random permutation of the items are fed to the first-fit heuristic. At the end of the initialization it is ensured that the capacity of each bin is not exceeded.

### 6.2. Fitness Function

A straightforward fitness function would be just taking the inverse of the number of bins. However, as pointed by Falkenauer [61] as well, such a fitness function will result an unfriendly fitness landscape in which many combinations with one more bin than optimal solution will have the same fitness value. Instead, the function proposed by [61] is used in this study:

$$f(s) = \frac{\sum_{i=1}^N (F_i/C)^k}{N} \quad (6.1)$$

In this formula,  $N$  is the number of bins,  $F_i$  is the fill of bin  $i$  and  $C$  is the maximum bin capacity, while  $k$  is the parameter of importance of well-filled bins. Setting  $k$  to 1 is the same as using the inverse of the number of bins while increasing  $k$  above 1 gives a higher fitness value to solutions comprised of more well-filled bins. Falkenauer reports that 2 is the optimal number for  $k$ .



### 6.3. Selection

A standart tournament selection with tournament size  $t$  is used in this study. It is observed that using higher selection pressure yields better result.

### 6.4. Crossover

As a crossover operator, mainly traditional one point crossover is adopted. After one point crossover, the resulting chromosome may violate the rules of LLE therefore a repair mechanism described in chapter 4 is applied.

---

**Require:** Two Parents -  $parent1$  and  $parent2$  in LLE form.

**Ensure:** Two *offspring* in LLE form.

```

1: for  $i \leftarrow 1$  to  $listSize$  do
2:    $endingParent1 \leftarrow \text{FindEndingNode of } value[i] \text{ in } parent1$ 
3:    $endingParent2 \leftarrow \text{FindEndingNode of } value[i] \text{ in } parent2$ 
4:    $whichParent \leftarrow \text{random}(TRUE, FALSE)$ 
5:   if  $whichParent = parent1$  then
6:      $value[i] \text{ in } child1 \leftarrow endingParent1$ 
7:      $value[i] \text{ in } child2 \leftarrow endingParent2$ 
8:   else
9:      $value[i] \text{ in } child1 \leftarrow endingParent2$ 
10:     $value[i] \text{ in } child2 \leftarrow endingParent1$ 
11:  end if
12: end for
13:  $\text{Repair}(child1)$ 
14:  $\text{Repair}(child2)$ 

```

---

Figure 6.1. Pseudocode of the modified uniform crossover

Apart from one point crossover, LIMX described in graph coloring algorithm and traditional uniform crossover were also used in the tests. A modified uniform crossover method

described in [63] to be used in clustering problem with LLE has also been tested. In the modified uniform crossover (MUX), instead of the actual values like in uniform crossover, the value of the ending node of the group in which the item belongs to is passed to the offspring. This ensures groups were not separated. It is observed that this crossover tends to combine the groups whose elements have the same ending node. The pseudocode of the MUX is presented in Figure 6.1.

After crossovers are applied a repair mechanism to eliminate double links to a node is necessary. The repair mechanism described in Section 4.3 is used.

### 6.5. First Fit Decreasing Rearrangement

Due to the nature of the crossover operators, the resulting individual may have bins whose capacities are exceeded and thus may need an additional repair procedure apart from the usual LLE repair mechanism. This repair procedure checks all of the bins and remove randomly selected items from the over-filled bins until the capacity is not exceeded anymore. The removed items are reinserted into the rest of the bins using a first fit heuristic. First fit decreasing, a variant of first fit in which items are sorted in decreasing order of size before insertion, has also been used. Both first fit and first fit decreasing can be implemented to use  $O(n \lg n)$  time where  $n$  is the number of items.

Like previous algorithms of Falkenauer [12] and reduction algorithm of Martello and Toth [43], a procedure based on the domination criterion has been adopted. When an excess item is to be inserted back to the solution, it is compared with the items already present in the bins first. An excess item replaces an item in the bin while not causing an overflow in the bin and the replaced item is put into the excess items list. This procedure helps to create more well-filled bins.

In all first fit rearrangement heuristics, lowest-index based ordering which is described in Chapter 5 is used. Due to the nature of the bin packing instances, cardinality based ordering will be no different random ordering therefore it is not used.

## 6.6. Mutation

In mutation, randomly chosen  $k$  bins are destroyed and contents of these bins are redistributed to rest of the bins using first fit heuristic. In our experiments we have observed that the performance of the algorithm is closely related with the parameter  $k$ , the number of bins that should be removed. Different values of  $k$  have been tested.

## 6.7. Replacement

For a replacement method, a simple trans generational genetic algorithm with weak elitism is used. Best  $n$  individuals with the best fitness value are preserved and other individuals are replaced with the offspring.

## 6.8. Experimental Results

In bin packing tests, two sets of test instances provided by Falkenauer [12] are used. In the first set, the maximum bin capacity is set to 150 and each integer item is randomly generated from a uniform distribution between 20 and 100. Falkenauer reports that the results of the Martello and Toth [43] reveal this distribution gives most difficult problems for their method. Falkenauer generated instances of this kind with the number of items 120, 250, 500 and 1000 with 20 instances each.

In the second set, the item sizes are drawn from the range  $(0.25, 0.50)$  to be packed into bins of maximum capacity 1. In these instances, a well-filled bin must contain one large item and two small items that is why Falkenauer referred them as 'triplets'. Even though it is possible to pack two large items or three small items into one bin, this results an inevitable loss of space. [12] points out a similarity between triplets and  $3SAT$  which is considered as the most difficult  $kSAT$  problem. Falkenauer finally points out that bin packing instances are easier to approximate when the number of items to packed into a bin exceeds three, so 'triplets' are the most difficult bin packing instances.

In order to preserve the difficulty of the problem, generated instances have known

local optima with maximum bin capacities of 1000 and a large item size  $s_l$  is randomly drawn uniformly from the range (380, 490). A small item size  $s_{s1}$  is drawn uniformly from the range. Finally a second small item size  $s_{s2}$  is set as  $1000 - s_l - s_{s1}$ . Triplets of 60, 120, 249 and 501 items with 20 instances each are then generated.

The general test setup is given on Table 6.1:

Table 6.1. Test setup for bin packing instances

<b>Test Machine:</b>	Pentium 4 2Ghz with 256MB Ram
<b>Compiler:</b>	GCC C++ 3.2 with -O2 flags
<b>Max Number of Generations:</b>	500
<b>Population Size:</b>	100
<b>Tournament Size:</b>	10
<b>Crossover Rate:</b>	1.00
<b>Mutation Rate:</b>	1, 2 or 4 bins are destroyed at random
<b>Repair Mechanism:</b>	First Fit (FF) and First Fit Decreasing (FFD)
<b>Number of Runs:</b>	1 for each instance in the set (total 20 runs)

For consistency with the previous experiments of Falkenauer [12] and Martello [43], each instance in one set is tested only once. This is because all instances in one set has the same characteristics of distribution and number of items.

6 sets of tests are carried out. The performance of four crossover operators (one point crossover - 1PTX, lowest index max crossover - LIMX, uniform crossover - UX and modified uniform crossover - MUX) are tested using first fit (FF) and first fit decreasing (FFD) heuristic. The mutation rate (the number of bins destroyed) is varied as 1, 2 and 4. In order to compare in a fixed heuristic and mutation rate setting, a ranking mechanism is implemented. Ranking method takes the success ratio (number of times the optimal solution is found), the mean number of bins found, and the mean number of generations into consideration in that order. T-tests are also carried out to ascertain if there are statistically significant differences

between different settings. If one crossover is better than all the others then it is given a ranking of 1, and if it is worse than all others, it is given a ranking of 4. In case of a tie rankings are shared. For example, if there is a tie in second and third positions, both crossovers get a score of 2.5. Obviously, a crossover is better if it receives a lower ranking. The rankings are presented in Tables 6.2, 6.3 and 6.4.

Rankings tests show that one point crossover is clearly the winner in all test setups. Modified uniform crossover is placed into the second position and is really an improvement over the regular uniform crossover which is placed third. Finally, lowest index max crossover which is designed specially for graph coloring comes the last.

Table 6.2. Rankings for first fit (decreasing) - 1 bin deleted

(a) First Fit Decreasing 1 Bin Deleted					(b) First Fit 1 Bin Deleted				
Inst.	1PTX	LIMX	UX	MUX	Inst.	1PTX	LIMX	UX	MUX
U120	2.0	4.0	2.0	2.0	U120	1.0	4.0	2.5	2.5
U250	3.0	4.0	2.0	1.0	U250	2.5	2.5	4.0	1.0
U500	1.5	3.0	4.0	1.5	U500	1.0	3.5	3.5	2.0
U1000	1.0	4.0	3.0	2.0	U1000	1.0	4.0	3.0	2.0
T60	2.5	2.5	2.5	2.5	T60	2.5	2.5	2.5	2.5
T120	2.5	2.5	2.5	2.5	T120	2.5	2.5	2.5	2.5
T250	2.5	2.5	2.5	2.5	T250	2.5	2.5	2.5	2.5
T500	1.5	1.5	3.5	3.5	T500	1.0	4.0	2.5	2.5
Avg.	2.06	3.00	2.75	2.19	Avg.	1.75	3.19	2.88	2.19

Table 6.3. Rankings for first fit (decreasing) - 2 bins deleted

(a) First Fit Decreasing 2 Bins Deleted					(b) First Fit 2 Bins Deleted				
Inst.	1PTX	LIMX	UX	MUX	Inst.	1PTX	LIMX	UX	MUX
U120	2.5	2.5	2.5	2.5	U120	1.0	4.0	2.0	3.0
U250	4.0	2.5	2.5	1.0	U250	1.0	2.5	4.0	2.5
U500	2.0	4.0	3.0	1.0	U500	1.0	4.0	3.0	2.0
U1000	1.0	4.0	3.0	2.0	U1000	1.0	4.0	3.0	2.0
T60	1.0	3.0	3.0	3.0	T60	1.0	3.0	3.0	3.0
T120	2.5	2.5	2.5	2.5	T120	2.5	2.5	2.5	2.5
T250	2.5	2.5	2.5	2.5	T250	1.5	1.5	3.5	3.5
T500	1.5	3.0	4.0	1.5	T500	1.0	3.0	3.0	3.0
Avg.	2.00	3.00	2.88	2.00	Avg.	2.00	3.06	3.00	2.69

Table 6.4. Rankings for first fit (decreasing) - 4 bins deleted

(a) First Fit Decreasing 4 Bins Deleted					(b) First Fit 4 Bins Deleted				
Inst.	1PTX	LIMX	UX	MUX	Inst.	1PTX	LIMX	UX	MUX
U120	2.0	4.0	2.0	2.0	U120	1.0	2.5	4.0	2.5
U250	3.5	2.0	3.5	1.0	U250	1.0	2.5	4.0	2.5
U500	1.0	4.0	3.0	2.0	U500	1.5	4.0	3.0	1.5
U1000	1.0	4.0	3.0	2.0	U1000	1.0	4.0	3.0	2.0
T60	3.0	1.0	3.0	3.0	T60	2.5	2.5	2.5	2.5
T120	2.5	2.5	2.5	2.5	T120	2.5	2.5	2.5	2.5
T250	1.5	1.5	3.5	3.5	T250	2.5	2.5	2.5	2.5
T500	1.0	4.0	2.5	2.5	T500	1.0	4.0	3.0	2.0
Avg.	1.94	2.88	2.88	2.31	Avg.	1.71	3.06	3.06	2.25

In terms of mean number of bins and number of generations, one point crossover usually gives best results especially for the more difficult large triplet instances. Lowest index

max crossover performs somewhat inconsistently, it sometimes gives results close to one point crossover on some large triplet instances however lags far behind especially in the larger uniform distribution instances. Uniform crossover and modified uniform crossover operators yet behave consistently and yield an acceptable performance in all test setups.

Poor performance of LIMX can be attributed to the characteristics of the instances. LIMX and the other crossover operators in graph coloring are not particularly suitable for the bin packing instances tested because for all of the instances average number of items per group is quite small (2 to 3 for uniform distribution instances and 3 for triplet instances). Preservation of large groups is not important due to the small average size of the bins and the low epistasis between items. Because of small and low epistasis bins, 1PTX has very good performance as likelihood of destruction of well-filled bins would be lower.

The hill climber heuristics which ensure no bins are overcapacitated are also compared. In Tables 6.5, 6.6, 6.7, 6.8, the rankings for each heuristics when used with a specific crossover operator are presented. First Fit (FF) and First Fit Decreasing (FFD) heuristics in which 1, 2 and 4 bins are destroyed give a total of 6 combinations. The ranking mechanism is the same as the crossover ranking taking success ratio, mean number of bins, and mean number of generations into account. The rankings range from 1 to 6.

After the ranking process, it is observed that different crossovers give their best performance in different heuristics. One point crossover performed best in a first fit heuristic and mutation in which 2 bins are deleted for mutation (FF2). Lowest Index Max Crossover operated best when used with FF1 which is closely followed by FF4. For uniform crossover, FFD2 performed best while for modified uniform crossover FFD1 is best and closely followed by FFD2.

Table 6.5. Rankings for hill climbers in one point crossover

Instance Set	FFD1	FFD2	FFD4	FF1	FF2	FF4
U120	2.5	2.5	2.5	2.5	2.5	2.5
U250	5.0	5.0	2.5	5.0	1.0	2.5
U500	4.5	4.5	1.5	1.5	4.5	4.5
U1000	3.5	3.5	6.0	1.0	3.5	3.5
T60	4.5	1.5	4.5	4.5	1.5	4.5
T120	3.5	3.5	3.5	3.5	3.5	3.5
T249	3.5	3.5	3.5	3.5	3.5	3.5
T501	3.5	3.5	3.5	3.5	3.5	3.5
AVG	3.81	3.44	3.44	3.13	2.94	3.50

Table 6.6. Rankings for hill climbers in lowest index max crossover

Instance Set	FFD1	FFD2	FFD4	FF1	FF2	FF4
U120	5.5	5.5	1.5	3.5	3.5	1.5
U250	5.5	2.0	1.0	3.5	3.5	5.5
U500	1.0	3.0	6.0	2.0	4.5	4.5
U1000	3.5	3.5	3.5	3.5	3.5	3.5
T60	4.0	4.0	1.0	4.0	4.0	4.0
T120	3.5	3.5	3.5	3.5	3.5	3.5
T249	3.5	3.5	3.5	3.5	3.5	3.5
T501	1.5	3.5	5.5	1.5	3.5	5.5
AVG	3.50	3.56	3.19	3.13	3.69	3.94



Table 6.7. Rankings for hill climbers in uniform crossover

Instance Set	FFD1	FFD2	FFD4	FF1	FF2	FF4
U120	2.5	2.5	2.5	2.5	5.0	6.0
U250	2.0	2.0	2.0	4.5	4.5	6.0
U500	5.0	2.5	1.0	4.0	6.0	2.5
U1000	5.5	2.0	1.0	5.5	4.0	3.0
T60	3.5	3.5	3.5	3.5	3.5	3.5
T120	3.5	3.5	3.5	3.5	3.5	3.5
T249	2.5	2.5	6.0	2.5	5.0	2.5
T501	2.5	2.5	4.0	1.0	5.5	5.5
AVG	3.38	2.63	2.94	3.38	4.63	4.06

Table 6.8. Rankings for hill climbers in modified uniform crossover

Instance Set	FFD1	FFD2	FFD4	FF1	FF2	FF4
U120	2.5	6.0	2.5	2.5	2.5	5.0
U250	1.5	3.0	1.5	4.0	5.0	6.0
U500	2.5	1.0	5.0	6.0	4.0	2.5
U1000	5.0	2.0	1.0	6.0	4.0	3.0
T60	3.5	3.5	3.5	3.5	3.5	3.5
T120	3.5	3.5	3.5	3.5	3.5	3.5
T249	2.5	2.5	6.0	2.5	5.0	2.5
T501	1.5	1.5	4.5	4.5	4.5	4.5
AVG	2.81	2.88	3.44	4.06	4.00	3.81

Table 6.9 represents average number of bins obtained from the test instances for the best setups for each crossover (1PTX - FF2, LIMX - FF1, UX - FFD2, MUX - FFD1). The theoretical minimum lower bound on the number of bins is given on the *theo* column. The average number of bins obtained in each set of instances are provided in 1PTX - FF2, LIMX

- FF1, UX - FFD2, and MUX - FFD1 columns. For a comparison, Falkenauer's Hybrid Grouping Genetic Algorithm [12] and Martello and Toth's [43] results are represented on HGGA and M&T columns, respectively.

Table 6.9. Mean number of bins obtained in best setups for each crossover

Inst.	Theo	1PTX-FF2	LIMX-FF1	UX-FFD2	MUX-FFD1	HGGA	M&T
U120	49.05	49.05	49.10	49.05	49.05	49.15	49.15
U250	101.55	101.70	101.80	101.75	101.65	101.70	102.15
U500	201.20	201.30	202.35	201.50	201.30	201.20	203.40
U1000	400.55	400.65	417.50	401.45	401.05	400.55	404.45
T60	20.00	20.95	21.00	21.00	21.00	20.10	201.55
T120	40.00	41.00	41.00	41.00	41.00	40.00	44.10
T249	83.00	84.00	84.00	84.15	84.05	83.00	90.45
T501	167.00	168.00	168.85	169.80	169.20	167.00	181.85
AVG	132.79	133.33	135.70	133.71	133.54	132.84	159.64

The results clearly show that the proposed algorithm is superior to Martello and Toth's reduction algorithm in all set of instances. Falkenauer's HGGA remains however the best algorithm in terms of overall solution quality especially for the more difficult triplet instances. However, for uniform distribution instances, LLE with one point crossover is very competitive, in fact for smaller instances it provides some packings that cannot be found with HGGA. However for triplet instances, LLE with 1PTX is consistently one bin short of the optimal packing. Interestingly, this deficiency is not dependent on the number of bins since even for a small triplet instance of 60 items, the optimal packing was not found which suggests that LLE with one point crossover was likely to get stuck at a local optimum.

Additional experimental results of bin packing can be found in Appendix B.

## 7. CONCLUSIONS

In this thesis, the performance of LLE has been tested on two well known grouping problems: graph coloring and bin packing. Several crossover operators that can be used with LLE have been introduced. These crossovers are based on principles to break symmetries by using lowest index and cardinality based ordering. However, results show that as long as groups are ordered in a consistent way, the performance does not vary between different orderings. It is also observed that preservation of large independent sets in a graph coloring problem is quite important as can be observed from the low performance of lowest index first crossover which ignores this principle. At the end of the graph coloring experiments, lowest index max crossover (LIMX) and greedy partition crossovers (GPX) performed similarly. Considering that GPX is an integral part of the most successful genetic algorithm, LIMX can be viewed as a promising operator.

One deficiency of these graph coloring crossovers appears when used in a multi-objective framework: they tend to produce a lot of small sized independent color sets due to the removal of vertices from parents during crossover. Although some attempts have been made to redistribute the vertices in these small sized sets to other large sets, these attempts were not successful. One obvious future research direction would be implementing a better reallocation method. This might increase the performance of the crossovers.

The traditional crossovers were never successful in graph coloring when used in LLE. This can easily be attributed to the fact that traditional crossovers destroy the independent sets which are the building blocks in graph coloring.

Unlike graph coloring, the performance of the traditional crossovers shine when used in bin packing. They were able to generate very competitive results close to hybrid grouping genetic algorithm (HGGA) of Falkenauer [12]. They match HGGA in uniform distribution instances and for smaller instances outperform them. However, for most difficult triplet instances, one point crossover, the best performing crossover in the test setup, was constantly one bin short of the optimal solution. This problem probably requires enhancements in the

mutation operator and this is one of the future research directions.

It is not expected that graph coloring crossovers to perform competitively, since in bin packing it is not crucial to preserve very large sets during generations. The most difficult test instances usually require packing few items to a bin (2 to 3 in this case). This is why ordering of groups based on cardinality does not make sense as it will essentially be no different than random ordering. These facts also explain why traditional crossover operators perform quite well in bin packing.

Linear linkage encoding is a viable candidate for solving grouping problems especially if the number of groups is not known beforehand. In such problems, the search is performed on a smaller search space than other encodings such as number encoding. In the future, operators that will make better use of this representation awaits research. Other grouping problem algorithms may utilize LLE as their representation method as well.

## APPENDIX A: EXPERIMENTAL RESULTS - GRAPH COLORING

Through Tables A.1, A.2, A.4 and A.4,  $\mu$  represents the mean of the best colorings obtained after 50 runs,  $\sigma$  stands for the standart deviation for this mean and B represents the best coloring of the 50 runs.

Table A.1. Average best, standard deviation and best results for the Carter's benchmark instances - Results for lowest index maximum and lowest index first crossovers

		LIMX			LIFX		
Instance	$\chi(G)$	$\mu$	$\sigma$	B	$\mu$	$\sigma$	B
Hecs92	?	17,66	0,47	17	17,86	0,4	17
Staf83	?	13,10	0,30	13	15,06	0,37	14
Yorf83	?	20,60	0,53	20	20,86	0,45	20
Utes92	?	10,00	0,00	10	10,02	0,14	10
Earf83	?	24,22	0,76	23	25,26	0,72	24
Tres92	?	22,26	0,52	21	22,32	0,51	21
Lsef91	?	19,20	0,66	17	19,10	0,67	18
Kfus93	?	20,86	0,57	20	21,30	0,61	20
Ryes93	?	23,86	0,49	23	23,96	0,66	23
Carf92	?	38,74	1,29	36	38,24	1,01	36
Utas92	?	40,52	1,03	38	40,84	1,01	39
Cars91	?	38,34	1,16	36	38,08	1,00	36

Table A.2. Average best, standard deviation and best results for the Carter's benchmark instances - Results for greedy partition lowest index and cardinality based crossovers

Instance	$\chi(G)$	GPX-LI			GPX-CB		
		$\mu$	$\sigma$	B	$\mu$	$\sigma$	B
Hecs92	?	17,80	0,40	17	17,76	0,43	17
Staf83	?	14,44	0,57	14	14,40	0,49	14
Yorf83	?	20,68	0,51	20	20,60	0,60	20
Utes92	?	10,00	0,00	10	10,00	0,00	10
Earf83	?	24,68	0,55	24	24,64	0,82	23
Tres92	?	22,26	0,56	21	22,30	0,57	21
Lsef91	?	19,16	0,67	18	19,32	0,58	18
Kfus93	?	21,12	0,65	20	21,20	0,57	20
Ryes93	?	23,90	0,64	23	23,68	0,55	23
Carf92	?	38,12	1,05	36	38,46	1,05	36
Utas92	?	40,60	1,08	38	40,88	1,01	38
Cars91	?	38,16	1,03	37	38,18	1,23	35

Table A.3. Average best, standard deviation and best results for the DIMACS Benchmark instances - Results for lowest index maximum and lowest index first crossovers

Instance	$\chi(G)$	LIMX			LIFX		
		$\mu$	$\sigma$	B	$\mu$	$\sigma$	B
DSJC125.5	?	18,70	0,46	18	18,76	0,43	18
DSJC125.9	?	44,66	0,47	44	44,74	0,48	44
zeroin.3.col	30	33,60	1,33	31	36,64	1,02	35
zeroin.1.col	49	49,04	0,20	49	50,66	0,55	50
zeroin.2.col	30	33,48	1,41	31	36,58	1,02	35
DSJC250.1	?	9,00	0,00	9	9,00	0,00	9
DSJC250.5	?	31,68	0,58	31	31,56	0,54	31
DSJC250.9	?	75,38	0,56	75	75,90	0,70	75
flat300_20	20	30,84	3,85	20	33,22	1,30	31
flat300_26	26	35,54	0,57	34	35,40	0,53	34
flat300_28	28	35,32	0,51	34	35,34	0,51	34
school1_nsh	14	14,82	0,82	14	14,82	0,89	14
le450_15a	15	16,98	0,32	16	17,00	0,20	16
le450_15b	15	16,82	0,43	16	16,98	0,14	16
le450_15c	15	23,82	0,52	23	23,70	0,46	23
le450_15d	15	23,94	0,47	23	23,66	0,51	23
le450_25a	25	25,00	0,00	25	25,10	0,30	25
le450_25b	25	25,00	0,00	25	25,08	0,27	25
le450_25c	25	29,08	0,34	28	29,24	0,43	29
le450_25d	25	29,12	0,43	28	29,20	0,45	28
DSJC500.1	?	14,00	0,00	14	14,16	0,37	14
DSJC500.5	?	55,98	0,55	55	55,96	0,49	55

Table A.4. Average best, standard deviation and best results for the DIMACS benchmark instances - Results for greedy partition lowest index and cardinality based crossovers

Instance	$\chi(G)$	GPX-LI			GPX-CB		
		$\mu$	$\sigma$	B	$\mu$	$\sigma$	B
DSJC125.5	?	18,66	0,47	18	18,68	0,47	18
DSJC125.9	?	44,24	0,43	44	44,36	0,48	44
zeroin.3.col	30	31,60	0,94	30	31,86	0,90	31
zeroin.1.col	49	49,00	0,00	49	49,00	0,00	49
zeroin.2.col	30	31,48	0,92	31	32,04	1,22	31
DSJC250.1	?	9,00	0,00	9	9,00	0,00	9
DSJC250.5	?	31,60	0,57	31	31,66	0,47	31
DSJC250.9	?	75,38	0,77	75	75,42	0,70	74
flat300_20	20	34,26	1,55	27	34,42	1,33	32
flat300_26	26	35,78	0,64	34	35,76	0,64	34
flat300_28	28	35,56	0,64	34	35,34	0,65	34
school1_nsh	14	14,60	0,80	14	14,94	0,95	14
le450_15a	15	16,92	0,27	16	16,96	0,20	16
le450_15b	15	16,92	0,27	16	16,90	0,30	16
le450_15c	15	23,76	0,43	23	23,58	0,49	23
le450_15d	15	23,78	0,50	23	23,58	0,53	23
le450_25a	25	25,00	0,00	25	25,02	0,00	25
le450_25b	25	25,00	0,00	25	25,00	0,00	25
le450_25c	25	29,08	0,34	28	29,02	0,32	28
le450_25d	25	29,12	0,47	28	29,08	0,34	28
DSJC500.1	?	14,00	0,00	14	14,00	0,00	14
DSJC500.5	?	56,00	0,57	55	56,18	0,71	55



## APPENDIX B: EXPERIMENTAL RESULTS - BIN PACKING

Through Tables B.1, B.2, B.3, B.4, B.5 and B.6, Best gives the optimal number of bins, mean represents the mean number of bins after 20 runs, fitness represents the mean fitness value of the best solutions and Gen. represents the mean number of generations.

Table B.1. Best packings obtained for the instances in the Falkenauer's benchmark suite -  
First fit decreasing with 1 bin deleted

		IPTX				LIMX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.977	3.70	1.00	49.15	0.974	80.40	0.90
U250	101.55	101.80	0.988	131.50	0.75	101.85	0.987	228.05	0.70
U500	201.20	201.30	0.994	84.25	0.90	203.10	0.979	377.00	0.70
U1000	400.55	400.65	0.997	93.95	0.90	417.60	0.923	500.00	0.00
T60	20.00	21.00	0.934	500.00	0.00	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.00	0.983	500.00	0.00	84.00	0.983	500.00	0.00
T501	167.00	168.00	0.992	500.00	0.00	168.55	0.985	500.00	0.00
AVG	132.79	133.35	0.979	289.18	0.44	135.78	0.966	398.18	0.29
		UX				MUX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.977	10.90	1.00	49.05	0.977	13.80	1.00
U250	101.55	101.75	0.988	176.50	0.80	101.65	0.990	131.60	0.90
U500	201.20	201.60	0.992	326.40	0.60	201.30	0.994	233.35	0.90
U1000	400.55	401.85	0.992	490.55	0.05	401.05	0.995	407.60	0.55
T60	20.00	21.00	0.934	500.00	0.00	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.00	0.982	500.00	0.00	84.05	0.982	500.00	0.00
T501	167.00	169.90	0.970	500.00	0.00	169.20	0.977	500.00	0.00
AVG	132.79	133.77	0.975	375.54	0.31	133.54	0.977	348.29	0.42

Table B.2. Best packings obtained for the instances in the Falkenauer's benchmark suite -  
First fit decreasing with 2 bins deleted

		1PTX				LIMX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.977	2.95	1.00	49.15	0.974	76.65	0.90
U250	101.55	101.80	0.988	129.05	0.75	101.75	0.988	220.60	0.80
U500	201.20	201.30	0.994	72.20	0.90	208.40	0.931	492.50	0.10
U1000	400.55	400.65	0.997	74.55	0.90	418.55	0.916	500.00	0.00
T60	20.00	20.95	0.938	477.10	0.05	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.00	0.984	500.00	0.00	84.00	0.983	500.00	0.00
T501	167.00	168.00	0.992	500.00	0.00	172.90	0.942	500.00	0.00
AVG	132.79	133.34	0.979	281.98	0.45	137.09	0.954	411.22	0.23
		UX				MUX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.977	15.05	1.00	49.10	0.976	30.95	0.95
U250	101.55	101.75	0.988	156.40	0.80	101.70	0.989	132.30	0.85
U500	201.20	201.50	0.992	288.90	0.70	201.25	0.994	163.35	0.95
U1000	400.55	401.45	0.993	447.00	0.40	400.80	0.996	308.65	0.75
T60	20.00	21.00	0.934	500.00	0.00	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.15	0.978	500.00	0.00	84.05	0.979	500.00	0.00
T501	167.00	169.80	0.971	500.00	0.00	169.35	0.975	500.00	0.00
AVG	132.79	133.71	0.975	363.42	0.36	133.53	0.976	329.41	0.44

Table B.3. Best packings obtained for the instances in the Falkenauer's benchmark suite -  
First fit decreasing with 4 bins deleted

		1PTX				LIMX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.978	2.45	1.00	49.05	0.977	27.95	1.00
U250	101.55	101.75	0.988	109.40	0.80	101.70	0.989	225.80	0.85
U500	201.20	201.25	0.994	40.15	0.95	209.85	0.917	500.00	0.00
U1000	400.55	400.70	0.997	94.45	0.85	418.15	0.917	500.00	0.00
T60	20.00	21.00	0.000	500.00	0.00	20.95	0.218	479.35	0.05
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.00	0.983	500.00	0.00	84.00	0.983	500.00	0.00
T501	167.00	168.00	0.991	500.00	0.00	177.50	0.896	500.00	0.00
AVG	132.79	133.34	0.862	280.81	0.45	137.78	0.858	404.14	0.24
		UX				MUX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.977	10.65	1.00	49.05	0.977	8.80	1.00
U250	101.55	101.75	0.988	159.50	0.80	101.65	0.989	119.30	0.90
U500	201.20	201.45	0.993	223.25	0.75	201.40	0.993	182.15	0.80
U1000	400.55	401.05	0.995	379.20	0.60	400.75	0.997	246.60	0.80
T60	20.00	21.00	0.000	500.00	0.00	21.00	0.000	500.00	0.00
T120	40.00	41.00	0.960	500.00	0.00	41.00	0.962	500.00	0.00
T249	83.00	84.65	0.966	500.00	0.00	84.70	0.966	500.00	0.00
T501	167.00	170.75	0.961	500.00	0.00	170.50	0.963	500.00	0.00
AVG	132.79	133.84	0.855	346.58	0.39	133.76	0.856	319.61	0.44

Table B.4. Best packings obtained for the instances in the Falkenauer's benchmark suite -  
First fit with 1 bin deleted

		IPTX				LIMX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.977	3.55	1.00	49.10	0.976	80.10	0.95
U250	101.55	101.80	0.988	138.70	0.75	101.80	0.988	226.20	0.75
U500	201.20	201.25	0.994	51.60	0.95	202.35	0.985	411.65	0.65
U1000	400.55	400.60	0.997	75.15	0.95	417.50	0.922	500.00	0.00
T60	20.00	21.00	0.934	500.00	0.00	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.00	0.983	500.00	0.00	84.00	0.983	500.00	0.00
T501	167.00	168.00	0.992	500.00	0.00	168.85	0.982	500.00	0.00
AVG	132.79	133.34	0.979	283.63	0.46	135.70	0.967	402.24	0.29
		UX				MUX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.977	28.10	1.00	49.05	0.977	27.35	1.00
U250	101.55	101.85	0.987	238.95	0.70	101.75	0.988	173.75	0.80
U500	201.20	201.60	0.991	359.45	0.65	201.40	0.993	230.90	0.80
U1000	400.55	402.45	0.989	485.00	0.05	401.25	0.995	427.00	0.35
T60	20.00	21.00	0.934	500.00	0.00	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.00	0.981	500.00	0.00	84.00	0.982	500.00	0.00
T501	167.00	172.75	0.940	500.00	0.00	170.55	0.963	500.00	0.00
AVG	132.79	134.21	0.971	388.94	0.30	133.75	0.975	357.38	0.37

Table B.5. Best packings obtained for the instances in the Falkenauer's benchmark suite -  
First fit with 2 bins deleted

		IPTX				LIMX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.977	4.50	1.00	49.10	0.976	79.25	0.95
U250	101.55	101.70	0.989	87.35	0.85	102.00	0.984	287.25	0.75
U500	201.20	201.30	0.994	62.95	0.90	209.55	0.921	493.35	0.05
U1000	400.55	400.65	0.997	68.70	0.90	418.55	0.916	500.00	0.00
T60	20.00	20.95	0.938	484.00	0.05	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.00	0.983	500.00	0.00	84.00	0.983	500.00	0.00
T501	167.00	168.00	0.992	500.00	0.00	173.20	0.939	500.00	0.00
AVG	132.79	133.33	0.979	275.94	0.46	137.30	0.952	419.98	0.22
		UX				MUX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.10	0.976	51.95	0.95	49.05	0.977	26.25	1.00
U250	101.55	101.85	0.987	228.30	0.70	101.80	0.988	178.75	0.75
U500	201.20	201.90	0.989	391.70	0.45	201.35	0.994	191.05	0.85
U1000	400.55	402.50	0.989	487.95	0.10	400.90	0.996	341.00	0.65
T60	20.00	21.00	0.934	500.00	0.00	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.25	0.975	500.00	0.00	84.30	0.975	500.00	0.00
T501	167.00	171.30	0.955	500.00	0.00	170.45	0.964	500.00	0.00
AVG	132.79	134.11	0.971	394.99	0.28	133.73	0.974	342.13	0.41

Table B.6. Best packings obtained for the instances in the Falkenauer's benchmark suite -  
First fit with 4 bins deleted

		IPTX				LIMX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.05	0.977	3.40	1.00	49.05	0.977	36.60	1.00
U250	101.55	101.75	0.988	107.00	0.80	102.00	0.984	264.75	0.70
U500	201.20	201.30	0.994	62.60	0.90	209.75	0.917	491.15	0.05
U1000	400.55	400.65	0.997	65.80	0.90	418.15	0.917	500.00	0.00
T60	20.00	21.00	0.934	500.00	0.00	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.00	0.983	500.00	0.00	84.00	0.983	500.00	0.00
T501	167.00	168.00	0.991	500.00	0.00	178.00	0.890	500.00	0.00
AVG	132.79	133.34	0.979	279.85	0.45	137.87	0.946	411.56	0.22
		UX				MUX			
Instance	Best	Mean	Fitness	Gen.	%	Mean	Fitness	Gen.	%
U120	49.05	49.15	0.974	82.95	0.90	49.05	0.977	47.75	1.00
U250	101.55	101.90	0.986	228.40	0.65	101.85	0.987	211.30	0.70
U500	201.20	201.50	0.992	265.60	0.70	201.30	0.994	145.50	0.90
U1000	400.55	402.05	0.991	462.80	0.25	400.85	0.996	304.35	0.70
T60	20.00	21.00	0.934	500.00	0.00	21.00	0.934	500.00	0.00
T120	40.00	41.00	0.966	500.00	0.00	41.00	0.966	500.00	0.00
T249	83.00	84.00	0.981	500.00	0.00	84.00	0.982	500.00	0.00
T501	167.00	172.75	0.940	500.00	0.00	170.55	0.963	500.00	0.00
AVG	132.79	134.17	0.971	379.97	0.31	133.70	0.975	338.61	0.41

## REFERENCES

1. Falkenauer, E., *Genetic Algorithms and Grouping Problems*, John Wiley and Sons, 1998.
2. Falkenauer, E., “*Applying Genetic Algorithms to Real-World Problems*”, L. D. Davis, K. De Jong, M. D. Vose and L. D. Whitley (Editors), *Evolutionary Algorithms*, pp. 65–88, Springer, New York, 1999.
3. Kirkpatrick, S., C. D. Gelatt and M. P. Vecchi, “*Optimization by Simulated Annealing*”, *Science, Number 4598, 13 May 1983*, Vol. 220, 4598, pp. 671–680, 1983.
4. Glover, F., “*Tabu Search, Part I*”, *ORSA Journal on Computing*, Vol. 1, pp. 190–206, 1989.
5. Holland, J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.
6. Darwin, C., *On the Origin of Species by Means of Natural Selection*, John Murray, London, 1859.
7. Du, J., E. Korkmaz, R. Alhadjj and K. Barker, “*Novel Clustering Approach that Employs Genetic Algorithm with New Representation Scheme and Multiple Objectives.*”, *In Proceedings of 6th International Conference on Data Warehousing and Knowledge Discovery - DaWaK '04, Zaragoza, Spain, 2004*.
8. Carter, M. W., G. Laporte and S. T. Lee, “*Examination Timetabling: Algorithmic Strategies and Applications.*”, *Journal of the Operational Research Society*, Vol. 47, pp. 373–383, 1996.
9. “*Cliques, Coloring and Satisfiability*”, Vol. 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society., 1996.
10. Galinier, P. and J. K. Hao, “*Hybrid Evolutionary Algorithms for Graph Coloring*”, *Jour-*

- nal of Combinatorial Optimization*, Vol. 3, No. 4, pp. 379–397, 1999.
11. Ramani, A., F. A. Aloul, I. Markov and K. A. Sakallah, “*Breaking Instance-Independent Symmetries in Exact Graph Coloring.*”, *Design Automation and Test Conference in Europe*, pp. 324–329, 2004.
  12. Falkenauer, E., “*A Hybrid Grouping Genetic Algorithm for Bin Packing*”, *Journal of Heuristics*, Vol. 2, pp. 5–30, 1996.
  13. Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freedman San Francisco, 1979.
  14. Avanthay, C., A. Hertz and N. Zufferey, “*Variable Neighborhood Search for Graph Coloring*”, *European Journal of Operational Research*, Vol. 151, pp. 379–388, 2003.
  15. Trick, M., *Network Resources for coloring a graph*, <http://mat.gsia.cmu.edu/COLOR/color.html>.
  16. Schaerf, A., “*A Survey of Automated Timetabling*”, *Artificial Intelligence Review*, Vol. 13, No. 2, pp. 87–127, 1999.
  17. Gamst, A., “*Some Lower Bounds for a Class of Frequency Assignment Problems*”, *IEEE Transactions of Vehicular Echnology*, Vol. 35, No. 1, pp. 8–14, 1986.
  18. Chaitin, G. J., “*Register Allocation & Spilling via Graph Coloring.*”, *SIGPLAN Symposium on Compiler Construction*, pp. 98–105, 1982.
  19. Briggs, P., K. D. Cooper and L. Torczon, “*Improvements to Graph Coloring Register Allocation*”, *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 3, pp. 428–455, May 1994.
  20. Chow, F. C. and J. L. Hennessy, “*The Priority-based Coloring Approach to Register Allocation*”, *ACM Trans. Program. Lang. Syst.*, Vol. 12, No. 4, pp. 501–536, 1990.



21. Garey, M. R., D. S. Johnson and H. C. So, “An Application of Graph Coloring to Printed Circuit Testing”, *IEEE Transactions on Circuits and Systems*, Vol. CAS-23, No. 591-599, 1976.
22. Klotz, W., *Graph Coloring Algorithms*, <http://pages.cpsc.ucalgary.ca/guox/gca.pdf>.
23. Breaz, D., “New Methods to Color Vertices of a Graph”, *Communications of the ACM*, Vol. 22, pp. 251–256, 1979.
24. Leighton, F. T., “A Graph Coloring Algorithm for large Scheduling Problems”, *Journal of Research of the National Bureau Standard*, Vol. 84, pp. 79–100, 1979.
25. Davis, L., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.
26. Hertz, A. and D. D. Werra, “Using Tabu Search Techniques for Graph Coloring”, *Computing*, Vol. 39, pp. 345–351, 1987.
27. Johnson, D. S., C. R. Aragon, L. A. McGeoch and C. Schevon, “Optimization by Simulated Annealing: An Experimental Evaluation: Part II, Graph Coloring and Number Partitioning”, *Operations Research*, Vol. 39, No. 3, pp. 378–406, 1991.
28. Dorne, R. and J. Hao, *Tabu Search for graph coloring, T-coloring and Set T-colorings*.
29. Wikipedia, *Simulated Annealing*, [http://en.wikipedia.org/wiki/Simulated\\_annealing](http://en.wikipedia.org/wiki/Simulated_annealing).
30. Fleurent, C. and J. A. Ferland, “Genetic and Hybrid Algorithms for Graph Coloring”, *Annals of Operations Research*, Vol. 63, pp. 437–461, 1996.
31. Glass, C. A. and A. Prügél-Bennett, “Genetic Algorithm for Graph Coloring: Exploration of Galinier and Hao’s Algorithm.”, *Journal of Combinatorial Optimization*, Vol. 7, No. 3, pp. 229–236, 2003.
32. Burke, E. K., J. Newall and R. F. Weare, “A Memetic Algorithm for University Exam Timetabling”, P. Burke, E.; Ross (Editor), *Practice and Theory of Automated*

- Timetabling, First International Conference 1995*, Lecture Notes in Computer Science 1153, pp. 241–250, Springer-Verlag, Berlin Heidelberg New York, August/September 1996.
33. Terashima-Marn, H., P. Ross and M. Valenzuela-Rendn, “*Clique-Based Crossover for Solving the Timetabling Problem with Gas*”, *Proceedings of the Congress on Evolutionary Computation*, pp. 1200–1206, 1999.
34. Paquete, L. F. and C. M. Fonseca, “*A Study of Examination Timetabling with Multiobjective Evolutionary Algorithms*”, *Proceedings of the 4th Metaheuristics International Conference*, pp. 149–154, Porto, 2001.
35. Wong, T., P. Cote and P. Gely, “*Final Exam Timetabling: A Practical Approach*”, *Canadian Conference on Electrical and Computer Engineering*, Vol. 2, pp. 726–731, Winnipeg, CA, 2002.
36. Ozcan, E. and E. Ersoy, “*Final Exam Scheduler - FES*”, *Proceedings of 2005 IEEE Congress on Evolutionary Computation*, Vol. 2, pp. 1356–1363, 2005.
37. Coffman, E. G., M. R. Garey and D. S. Johnson, “*Approximation Algorithms for Bin Packing: A Survey*”, pp. 46–93, 1997.
38. Garey, M. R., R. L. Graham and J. D. Ullman, “*Worst-case Analysis of Memory Allocation Algorithms*”, *STOC '72: Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, pp. 143–150, ACM Press, New York, NY, USA, 1972.
39. Wilson, P. R., M. S. Johnstone, M. Neely and D. Boles, “*Dynamic Storage Allocation: A Survey and Critical Review*”, *IWMM '95: Proceedings of the International Workshop on Memory Management*, pp. 1–116, Springer-Verlag, London, UK, 1995.
40. Brucker, P., *Scheduling Algorithms*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.
41. Coffman, E. G., M. R. Garey and D. S. Johnson, “*An Application of Bin-Packing to*

- Multiprocessor Scheduling.*”, *SIAM Journal on Computing*, Vol. 7, No. 1, pp. 1–17, 1978.
42. Dyckhoff, H., “*A Typology of Cutting and Packing Problems*”, *European Journal of Operational Research*, 1990.
  43. Martello, S. and P. Toth, “*Lower Bounds and Reduction Procedures for the Bin Packing Problem*”, *Discrete Applied Mathematics*, Vol. 28, No. 1, pp. 59–70, 1990.
  44. Wikipedia, *Genetic Algorithm*, [http://en.wikipedia.org/wiki/Genetic\\_algorithm](http://en.wikipedia.org/wiki/Genetic_algorithm).
  45. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, Reading, Massachusetts, 1989.
  46. Baker, J. E., “*Adaptive Selection Methods for Genetic Algorithms*”, *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 101–111, Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1985.
  47. Brindle, A., *Genetic algorithms for function optimization*, Ph.D. thesis, University of Alberta, Edmonton, Canada, 1981, unpublished.
  48. *Evolutionary Algorithms 1 Introduction*, [http://www.geatbx.com/ver\\_3\\_7/algindex.html](http://www.geatbx.com/ver_3_7/algindex.html).
  49. Moscato, P., *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*, Tech. Rep. C3P 826, Pasadena, CA, 1989.
  50. Baldwin, J. M., *Adaptive Individuals in Evolving Populations: Models and Algorithms - A New Factor in Evolution*, pp. 59–80, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.
  51. Whitley, D. L., V. S. Gordon and K. E. Mathias, “*Lamarckian Evolution, The Baldwin Effect and Function Optimization*”, Y. Davidor, H.-P. Schwefel and R. Männer (Editors), *Parallel Problem Solving from Nature – PPSN III*, pp. 6–15, Springer, Berlin, 1994.

52. Coello, C. A., “An Updated Survey of GA-based Multiobjective Optimization Techniques”, *ACM Computing Surveys*, Vol. 32, No. 2, pp. 109–143, 2000.
53. Deb, K., *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques - Multi-Objective Optimization*, Springer, 2005.
54. Srinivas, N. and K. Deb, “Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms”, *Evolutionary Computation*, Vol. 2, No. 3, pp. 221–248, 1994.
55. Fonseca, C. M. and P. J. Fleming, “Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization”, *Genetic Algorithms: Proceedings of the Fifth International Conference*, pp. 416–423, Morgan Kaufmann, 1993.
56. Horn, J., N. Nafpliotis and D. E. Goldberg, “A Niche Pareto Genetic Algorithm for Multiobjective Optimization”, *Proceedings of the First IEEE Conference on Evolutionary Computation*, Vol. 1 of *IEEE World Congress on Computational Intelligence*, pp. 82–87, Piscataway, New Jersey, 1994.
57. Crawford, J., M. L. Ginsberg, E. Luck and A. Roy, “Symmetry-Breaking Predicates for Search Problems”, L. C. Aiello, J. Doyle and S. Shapiro (Editors), *KR’96: Principles of Knowledge Representation and Reasoning*, pp. 148–159, Morgan Kaufmann, San Francisco, California, 1996.
58. Backofen, R. and S. Will, “Excluding Symmetries in Constraint-Based Search”, *Principles and Practice of Constraint Programming*, pp. 73–87, 1999.
59. Jones, D. R. and M. A. Beltramo, “Solving Partitioning Problems with Genetic Algorithms.”, *4th International Conference on Genetic Algorithms (ICGA)*, pp. 442–449, 1991.
60. Radcliffe, N. J., “Formal Analysis and Random Respectful Recombination”, *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 222–229, 1991.
61. Falkenauer, E. and A. Delchambre, “A Genetic Algorithm for Bin Packing and Line

- Balancing*”, *Proceedings of the IEEE 1992 International Conference on Robotics and Automation*, pp. 1186–1192, Nice, France, 1992.
62. Du, J., E. E. Korkmaz, R. Alhajj and K. Barker, “*Alternative Clustering by Utilizing Multi-objective Genetic Algorithm with Linked-List Based Chromosome Encoding.*”, *4th International Conference of Machine Learning and Data Mining in Pattern Recognition (MLDM)*, pp. 346–355, 2005.
63. Korkmaz, E. E., J. Du, R. Alhajj and K. Barker, “*Combining Advantages of New Chromosome Representation Scheme and Multi-objective Genetic Algorithms for Better Clustering*”, *Intelligent Data Analysis*, Vol. 10, No. 2, pp. 163–182, May 2006.
64. Ulker, O., E. Ozcan and E. E. Korkmaz, “*Linear Linkage Encoding in Grouping Problems: Applications on Graph Coloring and Timetabling*”, *International Conference on the Practice and Theory of Automated Timetabling*, 2006.
65. Feige, U. and J. Kilian, “*Zero Knowledge and the Chromatic Number*”, *IEEE Conference on Computational Complexity*, pp. 278–287, 1996.
66. Kirovski, D. and M. Potkonjak, “*Efficient Coloring of a Large Spectrum of Graphs*”, *35th Design Automation Conference Proceedings*, pp. 427–432, 1998.
67. Caramia, M., P. Dell’Olmo and G. F. Italiano, “*New algorithms for Examination Timetabling*”, *Algorithm Engineering 4th International Workshop 2000*, Lecture Notes in Computer Science 1982, pp. 230–241, Springer-Verlag, Berlin Heidelberg New York, September 2001.
68. Merlot, L. T. G., N. Boland, B. D. Hughes and P. J. Stuckey, “*A Hybrid Algorithm for the Examination Timetabling Problem.*”, P. Burke, E.; De Causmaecker (Editor), *Proceedings of Practice and Theory of Automated Timetabling, Fourth International Conference*, pp. 348–371, Gent, Belgium, August 2002.