ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

LOCAL SEARCH MANAGEMENT APPROACHES IN MEMETIC ALGORITHMS FOR SOLVING EXAM TIMETABLING PROBLEMS

M.Sc. Thesis by

Ersan ERSOY, B.Sc.

Department: **Advanced Technologies in Engineering**

Programme: **Computer Science**

JUNE 2007

İSTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

LOCAL SEARCH MANAGEMENT APPROACHES IN MEMETIC ALGORITHMS FOR SOLVING EXAM TIMETABLING PROBLEMS

M.Sc. Thesis by

Ersan ERSOY B.Sc.

704041007

Date of submission : 7 May 2007

Date of defence examination: 13 July 2007

Supervisor (Chairman): Asst.Prof.Dr. A. Şima UYAR

Co-Supervisor: Asst.Prof.Dr. Ender ÖZCAN (YÜ.)

Members of the Examining Committee Assoc.Prof.Dr. Haluk TOPÇUOĞLU (MÜ.)

Asst.Prof.Dr. E. Erkan KORKMAZ (YÜ.)

Asst.Prof.Dr. Osman Kaan EROL (İTÜ.)

JUNE 2007

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

SINAV ZAMANI ÇİZELGELEME PROBLEMLERİNİN ÇÖZÜMÜ İÇİN MEMETİK ALGORİTMALARDA YEREL ARAMA YÖNETİMİ YAKLAŞIMLARI

YÜKSEK LİSANS TEZİ

Müh. Ersan ERSOY

704041007

Tezin Enstitüye Verildiği Tarih : 7 Mayıs 2007
Tezin Savunulduğu Tarih : 13 Haziran 2007

Tez Danışmanı :	**Yrd.Doç.Dr. A. Şima Uyar**
Eş Danışmanı:	**Yrd.Doç.Dr. Ender Özcan (Y.Ü.)**
Diğer Jüri Üyeleri	**Doç.Dr. Haluk TOPÇUOĞLU (M.Ü.)**
	Yrd.Doç.Dr. E. Erkan KORKMAZ (Y.Ü.)
	Yrd.Doç.Dr. Osman Kaan EROL (İ.T.Ü.)

HAZİRAN 2007

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Asst. Prof. Dr A. Şima Uyar, and co-advisor Asst. Prof. Dr Ender Özcan for their valuable support, guidance and encouragement during my research.

Also, I would like to thank my family and Elif Gürbüz for their support, and patience.

I dedicated this thesis to the memory of my grandfather Kasım Erim.

Ersan Ersoy

May 2007

TABLE OF CONTENTS **Page**

LIST OF ABBREVITIATIONS **v**
LIST OF TABLES **vi**
LIST OF FIGURES **vii**
LIST OF SYMBOLS **viii**
ÖZET **ix**
SUMMARY **x**
1. INTRODUCTION **1**
2. TIMETABLING **3**
2.1 Examination Timetabling 4
2.1.1 Constraint Types of Examination Timetabling Problems 4
2.1.2 A Simple Problem Instance 5
2.1.3 Solution Approaches to Exam Timetabling Problems 6
2.1.3.1 Graph Based Approaches 7
2.1.3.2 Local Search Approaches 7
2.1.3.3 Constraint Based Approaches 8
2.1.3.4 Evolutionary Approaches 8
2.1.3.5 Multi-Criteria Approaches 9
2.1.3.6 Hyper-heuristics 9
3. PRELIMINARIES **10**
3.1 Genetic Algorithms 10
3.1.1 Parameters of Genetic Algorithms 14
3.1.2 Advantages and Disadvantages of Genetic Algorithms 14
3.2 Memetic Algorithms 15
3.3 Hyper-Heuristics 16
3.4 Ant Colony Optimization 19
4. MEMETIC ALGORITHMS FOR EXAM TIMETABLING **23**
5. EXPERIMENTAL RESULTS **28**
6. CONCLUSION AND FUTURE WORK **34**
REFERENCES **36**
BIOGRAPHY **40**

LIST OF ABBREVITIATIONS

GAs : Genetic Algorithms
MAs : Memetic Algorithms
EAs : Evolutionary Algorithms
SDF : Saturation Degree First Algorithm
LDF : Largest Degree First Algorithm
LWD : Largest Weighted Degree First Algorithm
GAs : Genetic Algorithms
MAs : Memetic Algorithms
EAs : Evolutionary Algorithms
ACO : Ant Colony Optimization
AS : Ant Systems
TSP : Traveling Salesman Problems
SR : Simple Random
CH : Choice Function
IE : Improved or Equal
GD : Great Deluge

LIST OF TABLES **Page**

Table 2.1: Students and their corresponding exams..... 6
Table 2.2: Possible solution of the sample problem 6
Table 5.1: Characteristics of experimental benchmark data 28
Table 5.2: Parameters of implemented MAs..... 28
Table 5.3. The performance comparison of the MAs with previously used approaches based on the best fitness values. The performance of seventeen approaches is ranked from 1 to 17, from the best towards the worst for each data and averages are given in the last column (Avr. ranks). 32

LIST OF FIGURES **Page**

Figure 2.1 : Illustration of the solution as a Graph Coloring Problem..... 6
Figure 3.1 : Procedure for Evolutionary Algorithms. 11
Figure 3.2 : Description of fitness based selection. 12
Figure 3.3 : One Point Crossover..... 14
Figure 3.4 : Illustration of hyper-heuristic framework. 17
Figure 3.5 : Sample Pseudo code for hyper-heuristics..... 17
Figure 3.6 : Pseudo code for hyper-heuristic framework with Great Deluge acceptance criteria [26]. 19
Figure 3.7 : Illustrating of ants' food finding behavior..... 20
Figure 3.8 : Pseudo code of Ant Colony Optimization meta-heuristic. 20
Figure 4.1 : Procedure of Hill Climber Process 25
Figure 4.2 : Pseudo-code of a heuristic template for timetabling 26
Figure 5.1: The average rank (between 1 and 12, from the best towards the worst) of each MA over the benchmark data, where the textured bars denote the MAs using LWD and the others using LDF for initialization. Vertical lines are the average standard deviation of the corresponding method over six problems. 29
Figure 5.2 : The average rank (between 1 and 6, from the best towards the worst) of each MA with a different hyperhill-climber for second group of experiments. Vertical lines are the average standard deviation of the corresponding method over six problems. 30
Figure 5.3 : The average rank (between 1 and 5, from the best towards the worst) of each MA with a different hyperhill-climber for third group of experiments. Vertical lines are the average standard deviation of the corresponding method over deviation of six problems..... 31
Figure 5.4 : The average rank (between 1 and 5, from the best towards the worst) of MAs with hyperhill-climbers of third group that use five hill climbers. Vertical lines are the average standard deviation of the corresponding method over six problems. 33

LIST OF SYMBOLS

V	: Variable Set of a Timetabling Problem.
D	: Domain Set of a Timetabling Problem.
C	: Conflict Set of a Timetabling Problem.

SINAV ZAMANI ÇİZELGELEME PROBLEMLERİNİN ÇÖZÜMÜ İÇİN MEMETİK ALGORİTMALARDA YEREL ARAMA YÖNETİMİ YAKLAŞIMLARI

ÖZET

Genetik Algoritmalar (GAs) ile yerel arama tekniklerini birleştiren Memetik Algoritmalar (MAs), NP-Tam olan sınav zamanlama problemlerinin çözümü için etkili metotlardır. Bununla beraber, çok kısıtlamalı sınav zamanlama problemlerinin çözümü için olan MAs çok sayıda yerel arama metodu içerebilir. Bu durumda, algoritmanın başarısı, bu metotların yönetilmesine bağlıdır. Farklı yöntemler, farklı kalitede sonuçlar doğurur. Bir kısıtlamanın ihlalinin düzeltilmesi, başka bir kısıtlama için ihlaller yaratabilir. Bu çalışmada, uygun bir tepe tırmanma yönetim mekanizmasının bulunması amaçlanmaktadır. 2 tip başlatma metodu, 16 farklı tipi içeren 3 farklı tepe tırmanıcı yönetim mekanizma grubu uygulanmıştır. Bu mekanizmalara *hipertepe-tırmanıcıları* olarak adlandırılmışlardır. Hipertepe-tırmanıcıları her biri farklı bir kısıtlamayı sağlamayı çalışan 3 farklı tepe tırmanıcısını kullanma yöntemine göre farklıdır. İlk grupta, tepe tırmanıcılar önceden belirlenmiş bir sırayla teker, teker uygulanırlar. İkinci grupta, tepe tırmanıcıların sıralanması için kısıtlamaların ihlal bilgisi kullanılır. Ek olarak, bir tanesi rasgele tepe tırmanıcılarını sıralayan ve diğeri de tepe tırmanıcıları yerine karınca sistemi kullanan iki MA bu gruba eklenmiştir. Son grubun yönetim metotları hiper-sezgisel yöntemleri kullanmaktadır. Bir hiper-keşifsel önce alt seviye keşifsel yöntemlerden bir tane keşifsel yöntem seçer ve aday çözüme uygular. Kabul etme kriterine göre elde edilen sonuç kabul edilir veya edilmez. Deney sonuçları hem kendi içlerinde hem de literatürde sunulan diğer tekniklerin sonuçlarıyla karşılaştırılmıştır. Deneyler gösterir ki, sınav zamanlama problemleri için olan Memetik Algoritmalarda tepe tırmanıcıların yönetimi için hiper-sezgisel stratejilerinin kullanımı daha iyi sonuçlar veriyor.

LOCAL SEARCH MANAGEMENT APPROACHES IN MEMETIC ALGORITHMS FOR SOLVING EXAM TIMETABLING PROBLEMS

SUMMARY

Memetic Algorithms (MAs), that combine Genetic Algorithms (GAs) with local search techniques, are effective methods for solving exam timetabling problems which are NP-complete. Furthermore, MAs for solving multi-constraint examination timetabling problems can have multiple local search methods. In this situation, success of the algorithm is depended on the management of these methods. Different policies are resulted in different quality of solutions. Repairing violations of one constraint can create violations for another constraint. In this study, finding a proper hill climbing management mechanism is aimed. Two types of initializations and three categories of hill climbing management mechanisms that consist of sixteen different types are implemented. These mechanisms are named as *hyperhill-climbers*. Hyperhill-climbers are different in policy of using three kinds of hill climbers; each one is responsible for satisfying different type of constraints. In the first group, hill climbers are applied one by one in a predetermined order. Violation information of constraints is used for ordering of the hill climbers in the second group. In addition, two MAs, one of which randomly make an apply order of hill climbers and the other executes an Ant System instead of hill climbers, are included into this group. Management methods of last group use hyper-heuristic policies. A hyper-heuristic select a heuristic from a set of low level heuristics and apply to a candidate solution. The obtained solution can be accepted or not according to an accepting criterion. Experimental results are compared within themselves and solutions of other techniques proposed in literature. Experiments show hyper-heuristic strategies for the management of hill climbers give better solutions in Memetic Algorithms for solving examination timetabling problems.

1. INTRODUCTION

A simple examination timetabling problem is an NP-complete problem which can be defined as assigning of exams to time slots. Given x exams and y time slots, search space contains x^y candidate solutions. Problem gets more complex as new constraints are added and cannot be solved in a polynomial time. As a result, instead of using traditional algorithms, some alternative methods, such as Memetic Algorithms (MAs), are used.

Memetic Algorithms embed local search techniques into Genetic Algorithms (GAs). In the case of multiple constraints, MAs can use multiple local search algorithms. Yet, applying a local search mechanism for solving a constraint can create violations for other constraints. Furthermore, finding a general mechanism for solving all problems is very challenging. In this situation, management of local search techniques becomes more significant.

In this thesis, methods for managing multiple hill climbers in MAs are studied. Implemented mechanisms are named as *hyperhill-climbers*. This study is one of the first detailed research work on this topic. Sixteen different mechanisms that are categorized into three groups are implemented. In the first group hill climbers are applied in a predetermined order. Management policies use violation information for ordering the hill climbers in the second group. Also, a MA, which applies the hill climbers in a random order, and another MA, which executes an Ant System instead of the hill climbers, are included in this group. Hyper-heuristic policies are used to manage hill climbers in the last group. A hyper-heuristic selects a heuristic from a set of low level heuristics and apply to the solution. The resulting solution can be accepted or not according to an accepting criterion.

Thesis starts with explaining timetable problems and solution approaches. In Section 3 GAs, MAs, Ant Systems and hyper-heuristics are described. In the next section, detailed

information about the implemented algorithms, and ideas behind the approaches are given. Thesis ends with experimental results, conclusion and future work.

2. TIMETABLING

Timetabling can be described as the assignment of resources to some specific domains according to the various types of constraints. Resources can be courses, exams, people or any other objects and domains may include time slots, rooms, and etc. Burke, Kingston and de Werra stated that "A timetabling problem is a problem with four parameters.: T , a finite set of times; R, a finite set of resources; M , a finite set of meetings; and C a finite set of constraints. The problem is to assign times and resources to the meetings so as to satisfy the constraints as far as possible." [1]. In our real world where the time is the most important value, we encounter with many types of these problems. For instance, educational timetables, nurse timetables, jury timetables, sport, and conference timetables are solutions of these problems.

Comparing to these timetabling problems, educational timetabling problems are one of the most examined type in the literature. Periodically great effort is made for obtaining an acceptable timetable. On the other hand, an acceptable solution is not enough in general. Since the solution affects different kinds of people, such as students, and lecturers, it has to cover extra expectations of these different roles, too.

Educational timetabling problems can be classified into 3 major types[2]:

- **School timetabling:** A weekly assignment of the classes of a high school without making any confliction between the classes of a teacher and satisfying other requirements.
- **Course timetabling:** A weekly assignment of the courses of a university according to some specific resources (lecturers, classrooms, etc.) without making any confliction between these resources and satisfying other requirements.
- **Examination timetabling:** Assignment of the exams of courses without making any confliction of a student's exams and satisfying other constraints.

Since it can not be solved by a deterministic polynomial algorithm, advanced search techniques are not guarantied to find an optimal solution, real world constraints are difficult to implement, and each instance of the problem may have specific constraints, these problems occupies the challenging parts of Artificial Intelligence techniques [1]-[4].

Examination Timetabling

An examination timetabling problem can be represented as a combination of sets (V, D, C) as described in [5], where V is the set of variables (exams, etc.), D is the set of the domain variables (time slots, rooms, etc.) and finally C is the set of constraints that are expected to be satisfied. Each member of V has to be assigned to a specific type domain value. The problem is then simply defined as assigning of each variable in set V to a specific value in set D to make a timetable in which constraints in set C are optimally satisfied.

2.1.1 Constraint Types of Examination Timetabling Problems

Complexity of solving timetable problems results from the various types of defined constraints. These constraints sometimes contradict with each other, and also sometimes they are impossible to be satisfied. In this situation goal is minimizing the occurrence of the violations.

Constraints in timetabling problems are classified into two main categories: Hard constraints, and soft constraints. Hard constraints must not be violated in any situation. Having no overlap between courses of a student or having adequate seat capacity at any time slot are examples of hard constraints. A solution, which satisfies all the hard constraints, is called a feasible solution. Furthermore, violations of the soft constraints must be minimized. Satisfying of soft constraints is not necessary, but pleasing. Setting a

predefined time to an exam or increasing the free time between each exam of a student are examples of soft constraints. Soft constraints are extra expectations and the number of soft constraints is much more than hard constraints'. Yet, quality of a timetable is determined by the satisfied ratio of soft constraints. [6]-[8].

Some common hard and soft constraints of exam timetabling problems are listed below:

- *Hard Constraints:*
  - Exams may be assigned to a specific set of domain values (time slots)
  - Resources at any time slot must be adequate (room capacity, room capability).
- *Soft Constraints:*
  - Timetable must be spread out.
  - Exams with common questions must be assigned to a same time slot.
  - Exams with largest number of students must be assigned in an early period.
  - Exams of a department must be assigned to rooms that are near the related department.
  - Rooms with large capacity must be firstly assigned.
  - Exams must be assigned in a determined order.
  - Exams must be assigned to a specific time slots.

2.1.2 A Simple Problem Instance

Suppose that our sample exam timetabling problem consists of five students with IDs 704041001, 704041002, 704041003, 704041004, 704041005, three exams (E1, E2, E3) and 2 days x 2 periods timetable. Students with their corresponding exams can be seen in Table 2.1. Our constraint set consists of one hard constraint, which each student has to take one exam at any time. A possible solution of this exam timetabling problem is given in Table 2.2. Another way of demonstrating the solution is illustrated in Figure 2.1. Problem solution is given as a solution of a graph coloring problem. Each vertex represents an exam and edges are the constraint violations between exams. In a graph coloring problem, goal is to give different colors to vertexes that combined with an edge. In the solution, E2 and E3 has same color and their color is different than color of E1 which means that E1 and E2 are assigned to the same time slot that is different than E2's

slot. This illustration shows that an exam timetabling problem which aims to satisfy no confliction can be represented as a graph coloring problem.

Table 2.1: Students and their corresponding exams.

Student	Exams
704041001	E1,E2
704041002	E3
704041003	E1,E3
704041004	E1,E2
704041005	E2

Table 2.2: Possible solution of the sample problem.

Period/Day	Day-1	Day-2
Period -1	E1	
Period -2		E2,E3

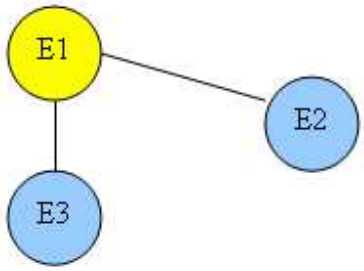


Figure 2.1 : Illustration of the solution as a Graph Coloring Problem.

2.1.3 Solution Approaches to Exam Timetabling Problems

Researcher has been developing different types of algorithms for solving examination timetabling problems since the last twenty years. These methods can be classified into six main types according to their major idea, which are graph based approaches, local search approaches, constraint based approaches, evolutionary approaches, multi-criteria approaches and hyper-heuristics [7]. Best results are obtained from the hybridization of these approaches.

2.1.3.1 Graph Based Approaches

Graph based algorithms are the first methods of solving exam timetabling problems. As it shown in Figure 2.1, an exam timetabling problem, where no confliction of exams of a student is aimed, can be modeled as a graph coloring problem in which vertices are the exams, and edges are the confliction between exams. Basic idea of graph coloring algorithms is ordering the exams and assigning to a time slot. Some ordering strategies are described below:

- **Saturation Degree First (SDF):** Ordering exams increasingly by the number of available time slots.
- **Largest Degree First (LDF):** Ordering exams decreasingly by the number of conflicts the exam contains.
- **Largest Weighted Degree First (LWD):** Ordering exams decreasingly by the number of conflicts the exam contains times number of students included.
- **Random Ordering:** Ordering exams randomly.
- **Largest Enrolment:** Ordering exams decreasingly by the number of enrollment time slots.

After ordering exams, assignments are done one by one according to most suitable time slot.

These strategies are quick and need less computational cost. Yet some early assignment can cause to stay at local optima. Because of this, they are effective approaches for obtaining initial solutions, which can be used with other algorithms [1][8].

2.1.3.2 Local Search Approaches

Local search techniques start solving the problem at a location in the search space, and visit to the neighborhoods of the current position. Selecting the next position is dependent on the move function and neighborhood structure. As a result, efficiency is

related to the moving function parameters and problem domain. Parameter tuning is an important pre-process of the local search algorithms [1].

- **Tabu Search:** Instead of directly moving on the search space, a tabu list is kept to prevent selecting visited points, which are not useful. Solutions worsen than the best solution points can also be selected to escape from the local optima. Efficiency of tabu search algorithm is also dependent to the problem instance and parameters, such as tabu list size.

- **Simulated Annealing:** It is derived from the natural annealing process in metallurgy. A temperature degree is determined at the beginning of the execution that is the probability of choosing worse points in the execution. This temperature decreases by a cooling factor. This process aims to search wide area of search space at the beginning of the execution and then focuses to the optimums at the end. Assigning of initial and final temperature and cooling factor parameters directly affects the performance.

2.1.3.3 Constraint Based Approaches

These types of algorithms model the timetabling problem as a set of variables and try to assign resources to these variables. Like expert system a set of rules are defined and used applied to make assignments. In a situation where any rule does not resulted in a feasible solution, a backtracking is performed until a solution is obtained that satisfies constraints [8].

2.1.3.4 Evolutionary Approaches

Evolutionary Algorithms consist of population based algorithms such as Genetic Algorithms (GAs), Memetic Algorithms (MAs) and Ant Algorithms These methods are very effective to obtain qualified solutions. GAs are derived from the evolution theory and consist of individuals called as chromosome. Each chromosome is a candidate solution to the problem. New chromosomes are obtained from two parent chromosomes by the help of genetic operators, such as crossover and mutation. By this way new candidate solutions are produced. In MAs local search techniques such as hill climbing or tabu search is combined with GAs. Ant Algorithms replicate the path finding

techniques of the natural ants. Algorithm consists of ants and each ant produce a solution by following a way brings some pheromone. In next cycle ants benefit from the pheromone value for choosing their way. By this way, better solutions are obtained. Evolutionary techniques are explained in Section 3.1 and Section 3.4 in detail.

2.1.3.5 Multi-Criteria Approaches

Real world timetabling problems consist of many constraints. Evaluating a single fitness value cannot guarantee dealing with each constraint. Multi-Criteria Algorithms proposes a vector of constraints' fitness values to be a solution of this situation. Each constraint is coped with individually and has its own importance. Value of importance of constraint can be changed during the execution [1].

2.1.3.6 Hyper-heuristics

Hyper-heuristics choose a heuristic from a set of heuristics and apply it to the current solution and accept or does not accept the result as the next generation solution. Effects of parameter tuning and the problem dependency cause to research new algorithms. Hyper-heuristics are proposed to be a problem independent, and aim to offer a general good solution instead of best of each problem instance. More information can be found in Section 3.3.

3. PRELIMINARIES

This section aims to explain in detail Genetic and Memetic Algorithms, Hyper-heuristics and Ant Algorithms, which are the algorithmic approaches implemented in the study.

3.1 Genetic Algorithms

Evolutionary Algorithms (EAs) are population based algorithms which are inspired from Darwinian Theory of evolution and population genetics. They are a member of stochastic search approaches and can be applied to many optimization problems with little information. Basic idea is based on the survival of the fittest and natural selection in which best individuals are chosen and to produce new generations while other candidates fade away and the population converges to optimum points.

Generational Evolutionary Algorithm procedure is described in Figure 3.1.

1. Set *genetationCounter* to 0 and randomly generate an initial population (*P(genetationCounter)*)
2. Do until termination break criteria is satisfied
   a. Evaluate the fitness of each individual
   b. Select parents of the next generation from *P(genetationCounter)* according to their fitness
   c. Generate new individuals by using search operators.
   d. Form new generation, *P(genetationCounter +1)* from new offspring pool and increment *genetationCounter* by 1.

Figure 3.1 : Procedure for Evolutionary Algorithms.

Genetic Algorithms (GAs) are a member of EAs in which main search operators are crossover and mutation. GAs was firstly proposed by Holland in 1970's [9]. In GA, candidate solutions are encoded as chromosomes, which form the individuals. Each individual is made up of genes, where each gene receives an allele from a set of predetermined values. For example, in a binary encoding an individual is a binary string, where {0,1} is the allele set. In the initializing phase, population can be generated randomly or by using algorithmic approaches. Some researchers benefit from pre-determined timetables for initialization [10]. A fitness function is used to measure the quality of an individual. An example fitness function is given in Equation 3.1:

$$\frac{1}{1+\sum_{i=0}^{N} w_i v_i} \qquad \textbf{(3.1)}$$

Where w_i is the weight of constraint i, and v_i is the total number of violations belongs to constraint i and N is the number of constraints.

In an evolutionary cycle, all individuals go through a set of genetic operations, i.e. selection, crossover and mutation. Depending on the fitness of all the individuals in the population, two of them, termed as mates, are randomly selected through a mechanism,

which favours individuals with better fitness values. There are three main types of selection strategy:

- **Fitness Based Selection:** A selection probability is given to all individuals according to their fitness value, and parents of new generation are chosen according to these probabilities. If the differences between the fitness values of chromosomes is huge, mostly best individuals are chosen for breeding, population consists of same gene values and a premature convergence will occur that means algorithm will be caught between the points in a local minimum. Fitness scaling techniques are used to avoid this problem. Pseudo code of fitness based selection is described in Figure 3.2 [12]

1. Calculate the sum of the fitness values of the population and assign to f_{sum}.
2. Take a random number, between 0 and f_{sum}.
3. Add the fitness of the chromosomes until the cumulative value becomes greater than random number.
4. Select the last individual as a parent.

Figure 3.2 : Description of fitness based selection.

- **Rank Based Selection:** In this approach, a selection probability is assigned to all individuals according to their rank. This method increases the diversity of the population by controlling the differences between the selection probabilities. On the other hand, this results in slow convergences to optima. Sample pseudo code is given below:

i) Assign selecting probability of each chromosome according to Equation 5.2

$$P_n = \frac{N_{pop} - n + 1}{\sum_{n=1}^{N_{pop}} n} \qquad \textbf{(3.2)}$$

Where N_{pop} is the size of population and P_n is the selecting probability.

ii) Take a random number between 0 and 1.

iii) Start adding the selecting probabilities and choose the chromosome as parent which has a cumulative selecting probability greater than the random number

- **Tournament Selection:** Randomly a subset of chromosomes, which has a size equal to the tournament size, are chosen and the one having the best quality as one of the mates is selected as parent. This method is the most similar to natural selection.

After selecting parent chromosomes, a crossover mechanism is applied to the selected mate, which results in generating new candidate solutions called the offspring. Individuals exchange their information in this state. One Point Crossover randomly determines a crossover location and exchanges the parts of the mates in the one side of this point forming two new individuals. In Uniform Crossover, each corresponding bit in the same place of the parent chromosomes can be swapped with a probability ratio. Higher ratios cause disruption and convergence becomes slow. Otherwise optimized values can be gathered very quickly. One point crossover mechanism is figured in Figure 3.3.

In the next step, mutation operator changes each allele to another value from the allele set, with a probability for each gene in each offspring. Finally, the current population is replaced using individuals from the current generation and from the offspring pool. In general new generation only consists of new offspring. In a steady-state GA, best n chromosomes from the old population can be included to new population. Evolution terminates whenever some criteria are satisfied, such as expected fitness or maximum number of generation. This operation prevents largest generation gap and computation time for calculating the fitness and other operations is not as much as trans-generational since these operations are not applied to whole population. But premature convergence is possible [12]-[15].

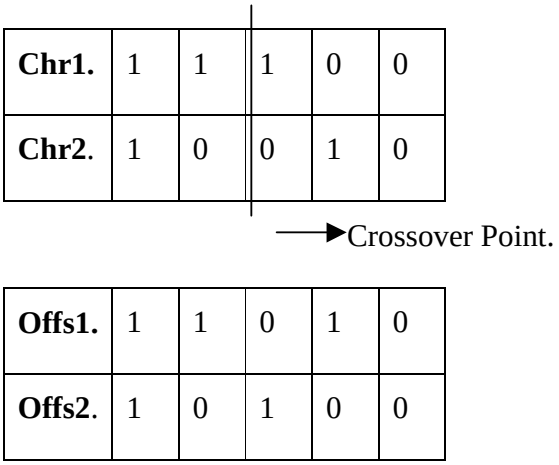


Figure 3.3 : One Point Crossover.

3.1.1 Parameters of Genetic Algorithms

Performance of GAs is directly related to the execution parameters. One of these parameters is crossover probability that shows how often crossover will be applied. Lower probabilities prevent the genetic information exchange of the individuals. Higher crossover probabilities are recommended in the literature [1]. Second operator is the mutation probability that determines the occurrence rate of mutation on a gene. By the help of mutation, GAs can jump over local minima. On the other hand, higher mutation rates can cause elitist chromosomes to be lost. Mutation rate can be adjusted, as one gene of each chromosome will be probably changed. Last parameter of GAs is the population size, which is equal to the number of chromosomes in the population. Assigning population size to a small value can cause to explore a small part of the search space and increases the probability of premature convergence. On the other hand, higher population size can be resulted in slow converging. Population size directly depended to the problem and increasing population size does not make any difference on the solution after a point.

3.1.2 Advantages and Disadvantages of Genetic Algorithms

GAs can be applied to various types of problems, such as optimization, machine learning, signal processing, pattern recognition, economics, etc. They are consistent to focus on local optimums. Also GAs do not need any information of the search space and

discontinuities of the search space has a little effect on GAs. In addition, GAs can be executed in parallel machines. On the other hand, GAs do not guarantee to find the optimum solution and needs high computational resources. Also tuning of parameter is a problem dependent and challenging pre-execution step.

3.2 Memetic Algorithms

Memetic Algorithms (MAs) are Evolutionary Algorithms in which local search techniques, such as hill climbing, and tabu search, are embedded into Genetic Algorithms to improve the exploitation capability of GAs. MAs were firstly introduced by Moscato and Norman [17]. MAs are inspired from the idea of meme, unit information adapted by each individual, and transferred to other individuals. The term meme is referred as the hill climbing strategy in MAs. MAs combine both local and global search approaches. (As a result applying MAs gives superior results than applying GAs.) Both exploration and exploitation is achieved. [18]. In a traditional MA, a hill climbing method is applied to each offspring following the mutation step. In this manner, a pool of improved offspring is formed. Notice that in case of the existence of a set of hill climbers, a mechanism can be used to select one from this set during the improvement stage without changing the original MA framework. For example, Krasnogor formalizes a co-evolutionary framework in [19] as multi-meme memetic algorithms and proposes a self-adaptive mechanism based on a Lamarckian learning approach.

3.3 Hyper-Heuristics

Hyper-heuristics, as defined by Cowling, Kendall and Soubeiga in 2000 [20], is a high level heuristic selection, applying and accepting mechanisms. Meta-heuristics and heuristic methods are problem dependent algorithms. Parameter tuning might take a major role in the success ratio of especially meta-heuristics. On the other hand, hyper-heuristics need simple details rather than problem domain knowledge. Meta-heuristics search the space of problem domain, while hyper-heuristics search the space of the heuristics. Hyper-heuristics aim to find the best heuristic or sequence of heuristics at a time to solve the problem at hand. A hyper-heuristic approach can be employed as a successful general method for solving different types of problems [20]-[24]:

Problem domain barriers separate problem dependent and independent parts in hyper-heuristic framework, Problem dependent part includes set of heuristics, and problem domain. On the other hand, problem independent part consists of hyper-heuristic mechanism. This is illustrated in Figure 3.4. Heuristics layers consist of mutational heuristics and hill climbers. Hill climbers aim to improve the candidate solution; on the other hand, mutational heuristics make random operation and does not guarantee an improved step. In a simple hyper-heuristic framework, firstly, a heuristics is chosen from the set of heuristics. In the second step, an acceptance mechanism is applied to decide if the output of the heuristic will be taken as a candidate solution in the next step. In this stage, input solution can be selected instead of the output. Execution terminates when a stopping criteria, such as obtaining expected fitness or iteration, occurs. General hyper-heuristic procedure is given in Figure 3.5. In addition to simple hyper-heuristic framework, Özcan, Bilgin and Korkmaz proposed three additional frameworks, in one of which improves the performance by applying a hill climber after selecting a mutational heuristics. [21]-[22].

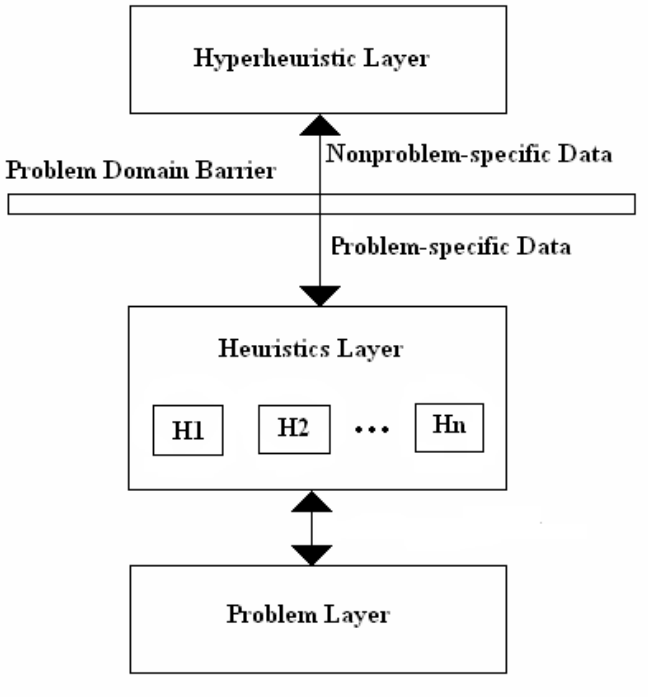


Figure 3.4 : Illustration of hyper-heuristic framework.

1. Generate an initial solution, named as c_i.
2. Calculate fitness of c_i by the help of objective function
3. If Stopping Criteria is not set.
   a. Select and apply a low level heuristic on c_i and called output as b_i.
   b. Calculate fitness of b_i.
   c. Apply acceptance mechanism to decide which solution will be used in the next step.
   d. If b_i is chosen.
      i. Set $c_i = b_i$.
   e. Go to step 2

Figure 3.5 : Sample Pseudo code for hyper-heuristics.

Efficiency of hyper-heuristics depends on selection and acceptance mechanisms. Various types of selection and acceptance mechanisms are implemented in the literature. In the early implementations of hyper-heuristics, Simple, Greedy, and Choice Function types of selection approaches are used [20]. Simple selection can be categorized into four types:

- **Simple Random:** Each time, a low level heuristics is chosen randomly.
- **Random Permutation:** An initially created permutation order of the low level heuristics is used. Each heuristics is applied one by one in each iteration.
- **Random Descent:** A low level heuristic is chosen and used until improving results is obtained.
- **Random Permutation Descent:** Same as Random Permutation but keeping using heuristic that produces improvements.

Another selection approach is Greedy selection in which all low level heuristics are applied and the best output is taken as the next candidate solution. Choice Function selection (CH), which makes an adaptation on the selection probability of the low level heuristics, is one of the most complex types of selection methods. Choice function benefits from the improvements of each heuristics, the improvements of each consecutive heuristics, and the execution time since the each low level heuristics was called. At the end of the iteration, parameters of Choice Function have to be updated.

A simple acceptance mechanism can be considered as accepting all moves of low-level heuristics (AM). Also only improving (OI: Only Improving) moves or moves, which does not produce worse solution (IE: Improving and Equal) can be accepted. Kendall and Mohamad implemented a Great Deluge (GD) acceptance criterion, in which all moves, generating a better or equal fitness value than a level computed at iteration, are accepted. The initial level can be set to the objective value of the initial candidate solution or the best value in the literature. At each step, the level is updated at a linear rate towards the expected objective value [26]. Their algorithm is illustrated in Figure 3.6.

1. Generate an initial solution, named as c_i.
2. Calculate fitness (f) of c_i by the help of objective function
3. Set initial *Level* = f.
4. Set *DownRate* = ($f - f_b$)/ *#_of_iteration* : f_b=Best result.
5. If Stopping Criteria is not set.
   a. Select and apply a low level heuristic on c_i and called output as c_n.
   b. Calculate fitness of c_n, and assign to f_n.
   c. If f_n < *Level* then Set $c_i = c_n$.
   d. Set *Level* = *Level* - *DownRate*
   e. Go to step 5

Figure 3.6 : Pseudo code for hyper-heuristic framework with Great Deluge acceptance criteria [26].

Another acceptance mechanism is called as Monte Carlo, in which all of the improving and some of the non-improving moves are accepted. In this approach, a probability value for acceptance of non-improving moves is calculated [26].

3.4 Ant Colony Optimization

Ant Colony Optimization (ACO), which is a class of population based algorithms, was firstly proposed by M. Dorigo and his colleagues in early 1990s [27]. ACO is derived from the behaviour of natural ants for finding and collecting in an optimal way. A natural ant with no idea of the place of food, randomly explores the environment. When it finds food, according to the quantity and quality of the source, it determines a level of chemical material, called as *pheromone*, and leave over the return way. Afterwards, ants can form an opinion about the place of food source by the help of pheromone quantity. Pheromone evaporates in the course of time, but it increases on the way of food, and its amount is more on the shorter ways, since ants can reach the source and bring food to the nest in a shorter time. As a result, ants will find an optimal way to the food source in

time. This situation is illustrated in Figure 3.7. Ants choose both Way-A and Way-B. Furthermore, ants, which prefer shorter way (Way-B), turn back to the nest in a shorter time, and use Way-B, again. As time goes on, pheromone amount will increase on Way-B, and on the other hand, number of ants choosing Way-A will decreases rapidly.

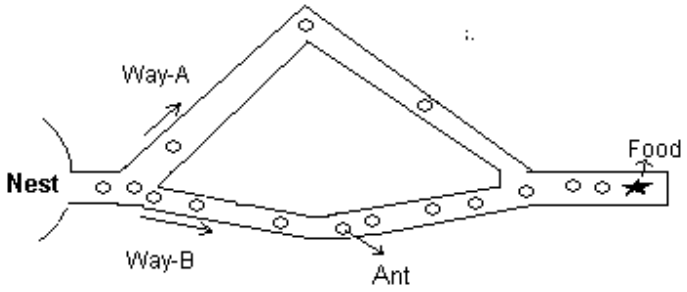


Figure 3.7 : Illustrating of ants' food finding behavior.

In the first approaches, referred to as Ant Systems (AS), Traveling Salesman Problems (TSP) are tried to be solved. Pseudo code of AS is described in Figure 3.8:

1. Initialize pheromone values.
   2. If Stopping Criteria is not set.
      a. Each ants generates a solution
      b. Update pheromone values
      c. Go to step 2

Figure 3.8 : Pseudo code of Ant Colony Optimization meta-heuristic.

A basic AS consists of equally powered computational agents, referred to as artificial ants. A trail table is stored for each state and is usually assigned to a specific value at the beginning. Ants move from one state i to another state j of the problem. This process can be considered as traveling from a city to another city in Traveling Salesman Problems. Ants decide the next state according to a probability, p_{ij}, computed from trail (τ_{ij}) ant attentiveness (η_{ij}) values. This probability is assigned to each available state by:

$$P_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha}.[\eta_{il}]^{\beta}}{\sum_{i \in Nij}[\tau_{ij}(t)]^{\alpha}.[\eta_{il}]^{\beta}} \quad \textbf{(3.3)}$$

Where Nij is the list of available states from state i, τ_{ij} is the trail amount and η_{ij} is the attractiveness when moving from state i to state j., α and β determine the effect of pheromone and heuristic information on choosing the next state.

Attractiveness, that determines the benefit of the state, is computed by a heuristic method. Updating of trails is called as evaporation and is applied when all ants finish their movements. Trails of states are decreased by an evaporation coefficient and increased with the pheromone values leaved by the ants. Decreasing process aims to prevent of reaching trails to infinite. Evaporation is formulated as:

$$\tau_{ij}(t+1)=(1-\rho).\tau_{ij}+\sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t) \quad \textbf{(3.4)}$$

Where ρ is evaporation coefficient which is in 0 and 1, τ_{ij} is the trail amount of the movement from state i to state j, $\Delta\tau_{ij}^{k}$ is the amount of pheromone that is leaved by ant k and computed according to the quality of the solution obtained same ant, m is the number of ants, and lastly t is the cycle number.

This cycle continues until a terminating criterion occurs.

There are different types of AS in the literature:

- **Elitist Ant System:** In this system, it is aimed to keep the information of the best solution. Because of this, pheromone trail of the states of the best solution is increased by an extra value.

- **Rank-Based Ant System:** This kind of systems sorts the solutions of each ant and permits ants which generate n best solutions to deposit a pheromone on the states. The parameter n is between one and the number of ants. If it is equal to the number of ants, algorithms behave like a classical AS:

- **MAX-MIN Ant System:** Ant that generates the best solution can deposit the pheromone trails in this system. There is an upper (τ_{max}) and lower limit (τ_{min}) to pheromone trails to prevent from stagnation. Trails are assigned to the upper limit initially. When a solution is found, or stagnation occurs, pheromone trails are reset to the upper limit.

ACO is a new, but rapidly improving approach. Many implementations on routing, assignment, scheduling, subset, and machine learning problems resulted in acceptable, high-quality solutions. Best results are obtained when local search techniques is embedded into ACO algorithms [27]-[29].

4. MEMETIC ALGORITHMS FOR EXAM TIMETABLING

Embedding local search to Genetic Algorithms makes a superior method, Memetic Algorithms. In combinatorial multi-objective problems, lots of constraints have to be satisfied. Only using one improving algorithm cannot be adequate to satisfy all constraints. As a result multiple improvement techniques, each one aiming to solve a different constraint, can be used in the local search phase. Therefore, management of these algorithms becomes significant. In this thesis, types of hill climbing management methods in Memetic Algorithms to solve an instance of exam timetabling problem are examined. Proposed management methods are referred to as hyperhill-climbers.

Derived methods in this thesis are tested on the exam timetabling problem provided by Carter et al. [31]. An optimum schedule for a set of exams is explored for spreading the timetable of the students. The cost function for evaluating the quality of a solution is computed as:

$$f(x) = \frac{1}{S}\sum_{i=1}^{E-1}\sum_{j=i+1}^{E} c_{ij} w_t \qquad \textit{where} \qquad w_t = \begin{cases} S * 10^6 & t = 0 \\ 2^{5-t} & t = 1,2,3,4,5 \\ 0 & t > 5 \end{cases} \tag{4.1}$$

Where E is the number of exams, S is the number of students, c_{ij} is the number of students taking both exams i and j; $(i, j \in (1 \ldots E))$, w_t is the violation weight for t free time slots between exams i and j.

A traditional Steady-State Memetic Algorithms with different hill climbing management techniques are implemented. Chromosome length is equal to the number of exams in the problem and each gene in the chromosome keeps the time slot value of a specific exam. The Order of an exam in the chromosome does not change Before the execution, an exam list with the number of students, confliction exam list of each exam, and confliction list are stored in memory for decreasing the computational cost. Initial

population can be generated in two ways. The first method, referred to as *Largest Degree First* (LDF), arranges the exams in a descending order in which the exam that causes the largest number of conflicts with the other exams is assigned first. During this process, one of the *available* periods is randomly assigned to the exam. In the assignment process, the time slots that do not cause a conflict are taken into an available time slot list and a random slot is chosen from them. If there is not any available slot, randomly a time slot from the timetable is assigned. The second method, referred to as *Largest Weighted Degree First* (LWD), schedules the exams in a similar way, Furthermore, number of students affected by the conflict is used instead of the raw number of conflicts. The assignment process is the same as LDF. Tournament selection with a tournament size of 4, one point crossover and traditional mutation is used. These operators give superior results [6].

Problem formulated in Equation (4.1) can be divided into three constraint types: C1, C2 and C3. C1 is a hard constraint, includes the violations due to the conflicting exams, which have to be assigned on a different time. C2 and C3 are soft constraints. C2 stands for spreading exams of a student with minimum three free slots, and this free slot number has to be six for C3. Based on this theory, three simple hill climbers are implemented to be used within the MA: HC1, HC2 and HC3. Each hill climber is responsible for satisfying the violations due to the corresponding constraint type.

After mutation, hyperhill-climbers use hill climbers (HC1, HC2 and HC3) to satisfy constraints. A Hill climber goes randomly through the relevant list of all exam pairs of each offspring once at a time by using confliction list. If a violation is detected, then a randomly selected exam is rescheduled to one of the available periods. If there is no such a period, then the other exam is tried to reschedule. After these operations, lists are updated. All hill climbers in this study operate in the same way. Figure 4.1 express this process in depth.

1. Randomly select each confliction list member one by one
2. If any violation of Hill Climber responsibility is set on exam A and B
   a. Randomly choose one exam. Suppose A is chosen.
   b. Remove slots which causes violations with A from available slots
   c. If any available slot remains
      i. Randomly assign slot from available slots
   d. Else remove slots which causes violations with B from available slots
   e. If any available slot remains
      i. Randomly assign slot from available slots

Figure 4.1 : Procedure of Hill Climber Process.

Implemented hyper–heuristics are organized in three groups. In the first group of mechanisms, six permutations of three hill climbers are determined and are applied to new offspring in the permutation order. The MAs using such a mechanism is symbolized by MA_123, MA_132, MA_213, MA_231, MA_312, and MA_321 where the order of the numbers {1,2,3} specifies the order of the sub hill climber. For instance, MA_123 applies HC1, HC2 and HC3 consecutively.

The second group of mechanisms orders the hill climbers based on the number of violations of each constraint type. Constraints are sorted according to the number of violations in descending order. Then the corresponding hill climbers are executed in that order. The *violation ordering hyperhill-climber* (VIOO) uses the number of violations for ordering the hill climbers and the *cost ordering hyperhill-climber* (CSTO) uses the weighted sum of violations instead of the number of violations. MAs that use these mechanisms are called as MA_VIOO and MA_CSTO respectively. A violation based heuristic ordering framework is proposed in [6]. In this thesis, a modified version of this approach, which is explained in Figure 4.2, is used. In this approach, first, selected hill climbers are applied to all violation events, unless any improvements cannot be gained. After this point, the area of concerns is lowered and hill climbers are applied to a randomly determined part of events. And finally, hill climbers are applied to one violation event. Selection of hill climbers can be done according to raw number (VIOD) or weighted sum of violations (CSTD). MA_VIOD and MA_CSTD are another two members of the second group hyperhill-climbers that use VIOD and CSTD in local

search step. Last two members of this group are MAs (MA_ANT and MA_RPG), that uses respectively an Ant System as local search technique and a Random Permutation Gradient like hyperhill-climber, which randomly orders HC1, HC2 and HC3 and applies all of them in each iteration. MA_ANT uses a MAX_MIN Ant System instead of HC1, HC2 or HC3. A pheromone trail table with a size of timetable is created for each exam. In detail, all available time slots of an exam have a pheromone trail. At initial, all pheromone trails except for the time slot determined in MA is assigned to a same value. Pheromone trails of time slots, which are assigned in MA, are set to a greater value. As a result, information from MA is transferred to AS. Attractiveness of each time slot of an exam is inversely proportional to the number of confliction due to the assigning exams to the slot. MAX-MIN Ant Systems is chosen because of fast convergence.

1. Mark the area of concern as all events
2. **While** (Some termination criteria are not satisfied) **do**
    a. **While** (There is improvement and some termination criteria$_2$ are not satisfied) **do**
        i. Select a constraint type after evaluating each constraint type violations for the marked events
        ii. Apply hill climbing for the selected constraint type to all events within the area of concern
    b. **End while**
    c. Lower the area of concern and mark the related events
3. **End while**

Figure 4.2 : Pseudo-code of a heuristic template for timetabling.

Last group of hyper hill climbers take use of hyper-heuristic strategies. MA_mGR randomly choose two sub hill climbers, applied each one separately to individual and accepts the most improved individual. Two selection methods (SR, CF) and two acceptance criteria (IE, GD) are used to form four Hyperhill-climbers, MA_SR_IE, MA_SR_GD, MA_CF_IE, MA_CF_GD. Each chromosome keeps the parameters of CH and parameters are randomly transferred to offspring in crossover. Updates are done after applying hyperhill-climbers. MAs that use GD keep a global level and each chromosome use this same value. In general this level is assigned to the fitness of the initial candidate at the beginning, but MAs are population-based techniques and initial average fitness of the initial population is assigned first.

Hill climbing mechanisms are applied with the aim to search equal number of candidate solutions at the end of the run. After the local search step, two best chromosomes from the old generation and all the new chromosomes are sorted according to their fitness values and best of them form the new generation.

In [30], parameter control techniques are classified based on: what is changed (operator prob-abilities, hill climbing method etc.), how the change is made (deterministic, adaptive, self-adaptive), the scope of the change (population level, individual level, etc.) and the evidence upon which change occurs (monitoring performance of operators, population diversity, etc.). In the deterministic method of changing the parameters, there is a deterministic rule, which is used to modify the parameters without using any feedback from the search. In the adaptive mechanisms, the feedback taken from the ongoing search guides the change in the parameters. In self-adaptation, the parameters are coded into the chromosomes and are allowed to evolve along with the individuals. The hyperhill-climbers used within the MAs during this study can be classified based on this terminology. The first group of hyperhill-climbers is deterministic, while the second group of mechanisms is adaptive mechanisms, except the deterministic one MA_RPG, and self-adaptive one MA_ANT. The other hyperhill-climbers in this group adaptively select an appropriate hill climber by dividing a candidate solution into three subparts based on a decomposable penalty oriented fitness function. Hence, a component level adaptation is employed. In the third group, MA_SR_GD is an adaptive mechanism, operating at individual level, while MA_CF_IE and MA_CF_GD are self-adaptive mechanisms, operating at population level. On the other hand, MA_mGR and MA_SR_IE are deterministic hyperhill-climbers.

5. EXPERIMENTAL RESULTS

In the experiments, six data sets of Carter's benchmark, which are described in Table 5.1, are used [31]. Parameters of experiments are fixed and presented in Table 5.2. 20 runs are performed for each of 28 different MA types. Parameters of Ant System are assigned to the values given in [29]. Ranking according to best fitness is used in the comparison of the mechanisms.

Table 5.1: Characteristics of experimental benchmark data.

Test Case	***No.of Exams***	***No. of Students***	***Enrollments***	***Density of Conflict***	***No. of Periods***
Hec-s-92	81	2823	10632	42.0%	18
Kfu-s-93	461	5349	25113	5.6%	20
Lse-f-91	381	2726	10918	6.3%	18
Sta-f-83	139	611	5751	14.4%	13
Ute-s-92	184	2750	11793	8.5%	10
Yor-f-83	181	941	6034	28.9%	21

Table 5.2: Parameters of implemented MAs.

Parameter Name	**Value**
Population Size	200
Crossover Rate	1
Mutation Rate	1 / (Chromosome Length)
Maximum Number of Generations	10000
Number Of Ants in MA_ANT	20
Number Of Cycles in MA_ANT	20
α in MA_ANT	1
β in MA_ANT	1
τ_{max} in MA_ANT	1
τ_{min} in MA_ANT	0
ρ in MA_ANT	0.7
Down Rate in GD	(Best Result – Avr.Initial fitness) / 10000

In the first part of the experiments, we examine the initialization methods and six predefined order of hill climbers. Each hyperhill-climber is tested with two initializing methods. Best results of each MA with different initializing method are ranked from 1 to 12, from the best towards the worst for each benchmark data and average ranks over the six benchmark problem is presented in Figure 5.1, in which the x-axis stands for the types of MAs and the y-axis denotes for the average ranks. Vertical bars are the average standard deviation of the associated MA over six problems. (Same illustration is used in comparing the results of other experiments, too.) Textured bars indicate the MAs that use LWD for initializing. Best results are obtained from MA_213 that applies LWD as the population initializing method. Another observation is that, using HC3 as the first hill climber resulted in poor solutions. Applying LWD for initializing population is more effective than applying LDF. Because of this, LWD is used in the next experiments.

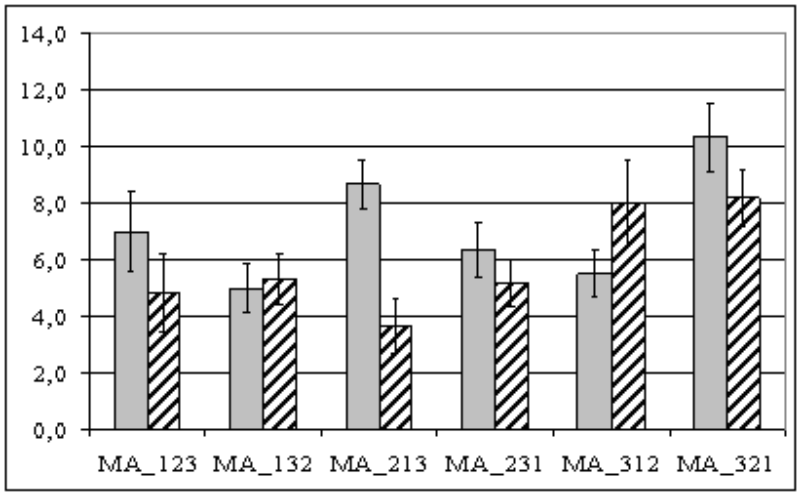


Figure 5.1 : The average rank (between 1 and 12, from the best towards the worst) of each MA over the benchmark data, where the textured bars denote the MAs using LWD and the others using LDF for initialization. Vertical lines are the average standard deviation of the corresponding method over six problems.

In the second set of experiments, MA_RPG, MA_ANT and violation based mechanisms are tested. Results are summarized in Figure 5.2. MA_CSTO provides the best solution over hyper-heuristics of this group. Applying hill climbers to a narrowing area of concern to some events seems to be an effective strategy when raw number of violations

is used for ordering the hill climbers. MA_VIOD has a better performance than MA_CSTD. Poorest results are obtained from MA_ANT in this phase. Ant colony optimization techniques are parametric methods. Tuning of the parameters determines the quality of the solutions, and the performance of the execution. Because of this, more experiments has to be done for MA_ANT with different parameter values. Probably, tuning parameters improve performance of MA_ANT.

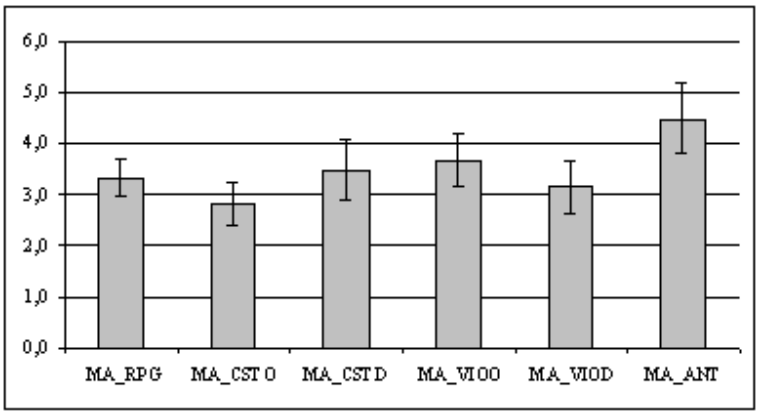


Figure 5.2 : The average rank (between 1 and 6, from the best towards the worst) of each MA with a different hyperhill-climber for second group of experiments. Vertical lines are the average standard deviation of the corresponding method over six problems.

MA_SR_IE and MA_CF_GD produce the best results in the last set of the experiments as it presented in Figure 5.3. Furthermore, MA_SR_IE has a smaller average standard deviation. In general, using IE gives better solutions. MA_mGR has the worst performance, although it visits more states as compared to the rest of the MAs in this group of experiments. The approach seems to get stuck at a local optimum due to the intensive use of hill climbers in MA_mGR.

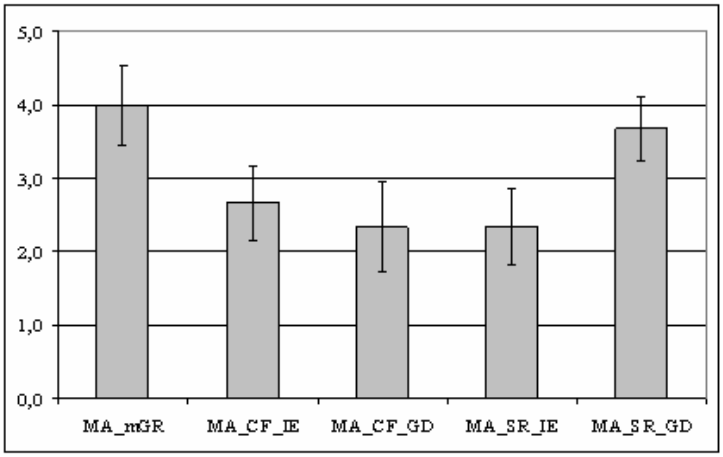


Figure 5.3 : The average rank (between 1 and 5, from the best towards the worst) of each MA with a different hyperhill-climber for third group of experiments. Vertical lines are the average standard deviation of the corresponding method over deviation of six problems.

Best MAs from each group are compared with the following previous approaches from the literature [29]:

- Largest Degree (LD), Saturation Degree (SD), Largest Weighted Degree (LWD) and Largest Enrollment (LE) rules of: Carter et al. [31].
- Tabu search approach (Wal) by White et al.[32].
- Tabu search approach (Gs) by Di Gaspero and Schaerf [33].
- Local search approach (Cal) by Caramia et al. [34].
- Great deluge local search approach (BN) by Burke and Newall [35].
- Simulated annealing approach (MAL) by Merlot et al. [36].
- Tabu search approach (PS) by Paquete and Stuetzle [37].
- Randomized adaptive search approach (CT) by Casey and Thomson [38].
- MAX-MIN Ant Algorithms (MMAS) by Michael [29].

The best MAs from the experiments and some previous approaches from the literature are compared. The approaches are divided into three groups: deterministic heuristics, stochastic methods, and the best MAs obtained during this study. Due to the parameter variances among the stochastic approaches and different termination criteria, the comparison between the algorithms can be considered as an indirect comparison as presented in Table 5.3. The hyperhill-climbers within the MAs provide a promising performance. MA_SR_IE turns out to be the best among the MAs, even though due to the termination criteria the maximum number of states it visits is fewer as compared to MA_213, MA_CSTO and MA_VIOD. MA_SR_IE and MA_CF_GD perform better than the LD, SD, LWD and LE heuristics.

Table 5.3. The performance comparison of the MAs with previously used approaches based on the best fitness values. The performance of seventeen approaches is ranked from 1 to 17, from the best towards the worst for each data and averages are given in the last column (Avr. ranks).

Approaches, [Source]	***Hec-s-92***	***Kfu-s-93***	***Lse-f-91***	***Sta-f-83***	***Ute-s-92***	***Yor-f-83***	***Avr. ranks***
LD	10.8	14.0	12.0	162.9	38.3	49.9	10,3
SD	12.7	15.9	12.9	165.7	31.5	44.8	12,6
LWD	15.8	22.1	13.1	161.5	26.7	41.7	12,5
LE	15.9	20.8	10.5	161.5	25.8	45.1	11,6
Wal	12.9	17.1	14.7	158.0	29.0	42.3	13,2
GS	12.4	18.0	15.5	161.0	29.9	41.0	13,5
Cal	9.2	13.8	9.6	158.2	24.4	36.2	2,6
BN	11.3	13.7	10.6	168.3	25.5	36.8	5,9
Mal	10.6	13.5	10.5	157.3	25.1	37.4	2,2
PS	10.8	16.5	13.2	158.1	27.8	38.9	8,6
CT	10.8	14.1	14.7	134.9	25.4	37.5	5,4
MMAS	11.3	15.0	12.1	157.2	27.7	39.6	6,4
MA_213	11.7	16.0	14.0	157.8	26.3	41.8	9,5
MA_CSTO	11.7	16.1	13.5	158.3	27.2	41.3	10,8
MA_VIOD	11.6	16.5	13.2	158.4	26.7	41.5	10,3
MA_CF_GD	11.8	16.1	13.4	157.7	26.3	40.9	8,8
MA_SR_IE	11.7	15.8	13.3	157.9	26.7	40.7	8,1

Last experiments of this study aim to examine the effect of the number of heuristics in hyperhill-climbers. Instead of three hill climbers (HC1, HC2, HC3) five low level hill climbers, each one is responsible to free time slot from 1 to 5, are used. Results are

given in Figure 5.4 Experiments show that SR selection produce poorer solutions when the number of low level heuristics increases, furthermore, choosing CH as selection strategy produce better performance. On the other hand, previous studies can not state that a selection method is superior [21][22].

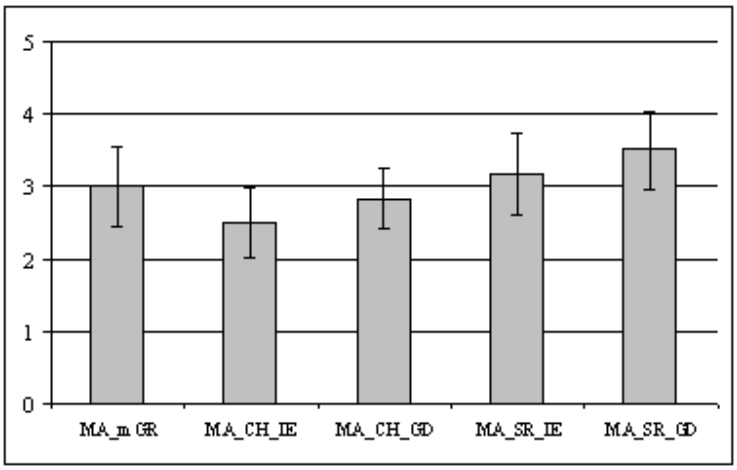


Figure 5.4 : The average rank (between 1 and 5, from the best towards the worst)of MAs with hyperhill-climbers of third group that use five hill climbers. Vertical lines are the average standard deviation of the corresponding method over six problems.

6. CONCLUSION AND FUTURE WORK

In this study, our purpose is to find a general hill climbing management mechanism for Memetic Algorithms (MAs) with multiple hill climbers. Exam timetabling problems are used to compare the performance of the derived algorithms. Three hill climbers (HC1, HC2, HC3), each aiming to reduce the violations of a specific constraint type are generated. Hill climbers management mechanisms, that are referred to as hyperhill-climbers, are categorized into three groups. In the first group, hill climbers are applied respectively in a predetermined order, permutations of three hill climbers. First group of methods are tested with two different initializing algorithms: Largest Degree First (LDF) and Largest Weighted Degree First (LWD). MAs that use LWD produce better solutions. Experiments show that hill climbers for satisfying the least important constraints have to be implemented at last.

In the second group of management mechanisms, violation information is used. Hill climbers are applied according to the order in which hill climber, which is responsible for the most violated cost, is applied first (MA_VIOD). Hill climbers can also use the weighted sum of the violations (MA_CSTO). Also heuristic ordering framework described in [6] is used to form two methods, MA_VIOD and MA_CSTD. These algorithms are compared with two other methods MA_RPG and MA_ANT. MA_RPG randomly order the hill climbers in each iteration. MA_ANT used a MAX-MIN Ant System in local search step. MA_ANT produces poorer solutions. Best results obtained from MA_CSTO.

In the last group of hyperhill-climbers, hyper-heuristic frameworks are embedded into Memetic Algorithms. Two selecting, Simple Random (SR), and Choice Function (CH), and 2 accepting mechanisms, Improving or Equal (IE) and Great Deluge (GD) are applied in Memetic Algorithms (MA_SR_IE, MA_SR_GD, MCH_IE, MA_CF_GD).

Another type of hyperhill climber in this group is MA_mGR which randomly chooses two of hill climbers, and applies them separately and takes best solution for the next generation. MA_SR_IE and MA_CF_GD produce the best performance.

The experimental results show that the hyperhill-climbers are viable strategies to manage a set of low level hill climbers within an MA. The self-adaptive hyperhill-climbers perform better as compared to the adaptive ones. Yet, a deterministic hyperhill-climber, namely, MA_SR_IE turns out to be the best one among all.

In the last set of experiments, we increase the number of hill climbers of hyperhill-climbers. Experiments show that SR selection method produce poorer solutions when the number of low level heuristics increases, furthermore, using CH as selection strategy produce better performance.

Hyperhill-climbers will be tested on more benchmark data; and parameter tuning techniques will be investigated in Ant System as future work.

REFERENCES

[1] **Qu, R., Burke, E. K., McCollum, B., Merlot, L. T . G., Lee, S. Y.,** 2006. A survey of Search Methodologies and Automated Approaches for Examination Timetabling, Technical report NOTTCS-TR-2006-4, Nottingham, Nottingham, UK.

[2] **Schaerf, A.**, 1996. A Survey of Automated Timetabling, Centrum voor Wiskund en Infirmatica (CWI), Technical report CS-R9567, Amsterdam, Netherlands.

[3] **Fang, H.,** 1994. Genetic Algorithms in Timetabling and Scheduling, *Ph.D. Thesis*, University of Edinburgh, Edinburgh.

[4] **Schaerf, A., Gaspero, Luca Di**,, 2001. Local Search Techniques for Educational Timetabling Problems, *Proceeding of the 6th International Symposium on Operational Research in Slovenia (SOR-'01)*, Preddvor, Slovenia, pp.13-23, September 30 – October 2.

[5] **Alkan,A., Özcan, E.**, 2003. Memetic Algorithms for Timetabling, *Proc. of 2003 IEEE Congress on Evolutionary Computation*, Canberra, Australia, pp. 1796-1802, December 8-12.

[6] **Özcan, E**., **Ersoy, E.**, 2005. Final Exam Scheduler, *Proc. of 2005 IEEE Congress on Evolutionary Computation*, pp. 1356-1363, Edinburgh, UK, September 2-5.

[7] **Burke, E., Elliman, D., Ford, P., and Weare, B.** 1996. Examination Timetabling in British Universities- A Survey. *Lecture Notes in Computer Science, Springer-Verlag*, vol. 1153 , pp.76–90.

[8] **Burke, E. K. and S. Petrovic,** 2002**.** Recent research directions in automated timetabling, *European Journal of Operational Research*, 140(2), pp.266-280.

[9] **Holland, J. H.**, 1975. Adaptation in Natural and Artificial Systems, Univ. Mich Press.

[10] **Burke, E.K., Newall, E, P., Weare F.**, 1995. A Memetic Algorithm for University Exam Timetabling, *Lecture Notes in Computer Science* 1153, pp. 241–250, Springer Verlag.

[11] **Wong, T., Côté P., Gely P.**, 2002. Final Exam Timetabling, *A Practical Approach* , École de technologie supérieure, Université du Quebec, Montréal Canada.

[12] **A Coley, D.**, 1999. An Introduction to Genetic Algorithms for Scientists and Engineers, World Scientific.

[13] **Areibi S:, Moussa M., Abdullah, H.**, 2001. A Comparison of Genetic/Memetic Algorithms and Other Heuristic Search Techniques, *2001 International Conference on Artificial Intelligence ICAI 2001*, Las Vegas, Nevada.

[14] **Radcliffe, N.J., Surry, P. D.**, 1994**.** Formal Memetic Algorithms, Lecture Notes in Computer Science Vol. 865, "Evolution-ary Computing", Springer Verlag, Berlin. pp. 1-16.

[15] **Ozcan, E.,** 2005. Memetic Algorithms for Nurse Rostering, *Lecture Notes in Computer Science, Springer-Verlag*, The 20th International Symposium on Computer and Information Sciences, vol. 3733, pp. 482-492.

[16] **L. Haupt, R., E. Haupt, S.,** 1998. *Practical Genetic Algorithms* John Wiley & Sons. Inc, New York.

[17] **Moscato P., Norman, M. G.,** 1992. A 'memetic' approach for the travelling salesman problem. Implementation of com-putational ecology on message passing systems. *Parallel Computing and Transputer Applications*, pp. 187-194, Amsterdam, Holland.

[18] **Radcliffe, N.J., Surry, P. D**., Formal Memetic Algorithms, *Lecture Notes in Computer Science* Vol. 865, Springer Verlag, Berlin. pp. 1-16.

[19] **Krasnogor, N.** 2002. Studies on the Theory and Design Space of Memetic Algorithms, *PhD Thesis*, University of the West of England, Bristol, UK.

[20] **Cowling P., Kendall, G., Soubeiga, E.**, 2000. A Hyperheuristic Approach to Scheduling a Sales Summit, *Proc. of the 3th International Conference on the Practice and Theory of Automated Timetabling*, Konstantz, Germany.

[21] **Özcan E., Bilgin B., Korkmaz E. E**., 2006. Hill Climbers and Mutational Heuristics in Hyperheuristics, The 9th International Conference on Parallel Problem Solving From Nature, Reykjavik, Iceland, Sept 9-13, pp. 202-211.

[22] **Özcan E., Bilgin B., Korkmaz E. E**., (to appear) A Comprehensive Analysis of Hyper-heuristics, *Intelligent Data Analysis*.

[23] **Kendall G., Cowling P., Soubeiga E.,** 2002. Choice Function and Random HyperHeuristics. *Proceedings of SEAL'02*, pp. 667-671.

[24] **Cowling P., Kendall G.,, Soubeiga E.,** 2001. A parameter-free hyperheuristic for scheduling a sales summit. *Proceedings of the 4th Metaheuristic International Conference, MIC 2001*, pp.127-131.

[25] **Ayob, M., Kendall, G.**, 2003. A Monte Carlo Hyper-Heuristic To Optimise Component Placement Sequencing For Multi Head Placement Machine. *Proceedings of the International Conference on Intelligent Technologies*, InTech'03, Chiang Mai, Thailand, Dec 17-19, pp.132-141.

[26] **Kendall G. Mohamad M.**, 2004. Channel Assignment in Cellular Communication Using a Great Deluge Hyperheuristic, *2004 IEEE International Conference on Network*, pp.769 – 773..

[27] **Colorni, A., Dorigo, M., Maniezzo, V.,** 1991**.** Distributed optimization by ant colonies, *Proceedings of ECAL'91, European Conference on Artificial Life*, Elsevier Publishing,

[28] **Dorigo, M., Stültzle., T**, 2004. Ant Colony Optimization, MIT Press.

[29] **Eley, M**., 2006. Ant algorithms for the exam timetabling problem. *Proc. of the 6th International Conference on the Practice and Theory of Automated Timetabling*, pp.167 – 180.

[30] **Eiben A. E., Hinterding R., Michalewicz Z.**, 1999. Parameter Control in Evolutionary Algorithms, *IEEE Trans. Evolutionary Computation, 3(2)*, 124 – 141.

[31] **Carter, M. W, Laporte, G., Lee, S. T.,** 1996, Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society*, 47, 373 – 383

[32] **White, G.M., Xie, B.S. and Zonjic, S.**, 2004 Using tabu search with long-term memory and relaxation to create examination timetables. *European Journal of Op. Research*, 153:80–91.

[33] **Di Gaspero, L. and Schaerf, A.**, 2001, Tabu search techniques for examination timetabling. *Lecture Notes in Computer Science*, Springer-Verlag, 2079:104–117.

[34] **Caramia, M. P., Dell'Olmo, and Italiano, G.F.** 2001. New algorithms for examination timetabling. *Lecture Notes in Computer Science*, Springer-Verlag, 982:230–241.

[35] **Burke, E.K. and Newall, J.**, 2003. Enhancing timetable solutions with local search methods. *Lecture Notes in Computer Science*, Springer-Verlag, 2740:195–206.

[36] **Merlot, L.T.G., Boland, N., Hughes, B.D. and Stuckey, P.J.** New benchmarks for examination timetabling. testproblem database, http://www.or.ms.unimelb.edu.au/timetabling.html.

[37] **Paquete, L. and Stuetzle, T.**, 2002. Empirical snalysis of tabu search for the lexicographic optimization of the examination timetabling problem. *Proc. of the 4th PATAT*, pp. 413–420, Berlin, Germany.

[38] **Casey, S. and Thompson, J.**, 2003. Grasping the examination scheduling problem. *Lecture Notes in Computer Science*, *Springer-Verlag*, 2740: 233–244.

BIOGRAPHY

Ersan Ersoy was born in Samsun, Türkiye on December 19, 1981. He completed his higher education at Samsun Anatolia High School. He received his Bachelors degree in Computer Engineering from Yeditepe University in 2004 and in the same year he started his graduate education Informatics Institute of Istanbul Technical University.

His professional interests comprise Evolutionary Algorithms, Meta-heuristics, Hyper-heuristics, Ant Colony Optimization, and Timetabling Problems.