T.C

BAHÇEŞEHİR UNIVERSITY

INSTITUTE OF SCIENCES

INDUSTRIAL ENGINEERING

SOLVING UNIVERSITY COURSE TIMETABLING PROBLEM USING GENETIC ALGORITHM

Master Thesis

M. ASLI AYDIN

İSTANBUL, 2008

T.C

BAHÇEŞEHİR UNIVERSITY

INSTITUTE OF SCIENCES

INDUSTRIAL ENGINEERING

SOLVING UNIVERSITY COURSE TIMETABLING PROBLEM USING GENETIC ALGORITHM

Master Thesis

M. Aslı AYDIN

Advisor: Asst. Prof. Dr. Ahmet BEŞKESE

İSTANBUL, 2008

T.C

BAHÇEŞEHİR UNIVERSITY

INSTITUTE OF SCIENCES

INDUSTRIAL ENGINEERING

Name of the thesis: Solving University Course Timetabling Problem Using Genetic Algorithm
Name/Last Name of the Student: M. Aslı AYDIN
Date of Thesis Defense: June.06.2008

The thesis has been approved by the Institute of Sciences.

Prof. Dr. Erol SEZER
Director

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

Asst. Prof. Dr. Ahmet BEŞKESE
Program Coordinator

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members	Signature
Asst. Prof. Dr. Ahmet BEŞKESE	_____
Prof. Dr. Cengiz KAHRAMAN	_____
Asst. Prof. Dr. F. Tunç BOZBURA	_____

ACKNOWLEDGMENTS

First of all I would like to thank to my family for always being there for me and for whatever I needed. And also I would like to thank to all those who have taken a part in my whole education.

Secondly I would like to thank to my supervisor Asst. Prof. Dr. Ahmet Beşkese for supporting this study by his guidance, help and time. Also, thanks to Asst. Prof. Dr. F. Tunç Bozbura whom gave me many valuable advises in the early stages of my thesis. And also I would like to thank to Assc. Prof. Dr. Irini Dimitriyadis for continously asking me "how is the thesis going?", because she encouraged me to persevere without knowing it.

Also I would like to thank to my best friend, my colleague, my dear husband Tarkan Aydın for his understanding, patience, encouragement, love and support. Also I thank him for his helpful discussions.

Finally, I would like to thank to Bahcesehir University for providing me the graduate scholarship.

June, 2008

M. Aslı AYDIN

ABSTRACT

SOLVING UNIVERSITY COURSE TIMETABLING PROBLEM USING GENETIC ALGORITMS

Aydın, Mürüvvet Aslı

Industrial Engineering

Advisor: Assoc. Prof. Dr. Ahmet Beşkese

June, 2008 37 pages

The university course timetabling problem consists of allocating a number of courses to a limited set of resources such as rooms, timeslots, set of lecturers and group of students in such a way to satisfy predefined constraints. The constraints can be divided into two groups: hard constraints and soft constraints. A timetable has to satisfy all hard constraints in order to be feasible and it should satisfy as much as possible soft constraints in order to be good quality.

The university timetabling problem is in class of NP-hard problems. This means that the amount of time and work required solving this type of problems increases exponentially with the problem size. This makes these problems more difficult and time consuming. Therefore optimization techniques are used to solve them and produce optimal or near optimal feasible solutions instead of exact solutions. Genetic algorithms are considered as an efficient approach for solving this type of problems.

Genetic algorithm is based on the principle of evolution first described by Charles Darwin and it tries to mimic some features of nature such as selection, crossover, mutation and replacement. Through these probabilistic operators, genetic algorithms choose the optimal solution among a set of alternate solutions which compete with each other.

In this thesis, first a general description of genetic algorithms and theoretical background, Schema Theorem, which describes the reasons of efficiency of genetic algorithms are given. Then a real life university course timetabling problem is examined with data coming from Bahcesehir University, Faculty of Arts and

Sciences. A solution based on genetic algorithm is proposed for the problem. The genetic algorithm implementations are described and applied to the sample problem. Moreover, determination of the best parameter values, such as the population size, mutation and crossover rate, etc. is tried. We work on this problem by introducing modified genetic operators which do not allow violations of any hard constraints, as a result produce only feasible solutions. Therefore the aim of the genetic algorithm is to produce a good quality timetable. This gains us to get better solutions in a comparative short time.

Keywords: Genetic Algorithms, Metaheuristics, Timetabling, Constraints

ÖZET

ÜNİVERSİTELERDE DERS PROGRAMI OLUŞTURMA PROBLEMİNİN GENETİK ALGORİTMA İLE ÇÖZÜMÜ

Endüstri Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Ahmet Beşkese

Haziran, 2008

Üniversite ders programı oluşturma problemi, belli bir sayıdaki derslerin sınırlı sayıdaki sınıf, zaman dilimi ve öğretim elemanları gibi kaynaklara önceden belirlenmiş kısıtları sağlayacak şekilde dağıtılmasını içerir. Kısıtlar iki gruba ayrılır: Kati kısıtlar ve gevşek kısıtlar. Bir ders programının kabil olabilmesi için bütün kati kısıtları sağlaması gerekir. İyi kalitede olabilmesi için de mümkün olduğunca fazla gevşek kısıtları sağlamalıdır.

Üniversite ders programı oluşturma problemi NP-zor problemler sınıfındadır. Bu şu demektir; bu tip soruları çözmede harcanan zaman ve yapılan iş, problemin büyüklüğü arttıkça eksponensiyel olarak artar. Bu da bu problemleri daha zor ve daha zaman alıcı yapar. Bu yüzden bunları çözmekte ve tam sonuç bulmak yerine en iyi veya en iyiye en yakın çözümler üretmek için eniyileme teknikleri kullanılır. Genetik algoritmalar bu tip problemlerin çözümünde verimli bir method olarak düşünülür.

Genetik algoritma, ilk kez Charles Darwin tarafından anlatılan evrim teorisini baz alır ve doğadaki seleksiyon, çaprazlama, mutasyon ve değiştirme gibi yapıları taklit etmeye çalışır. Bu olasılıksal operatörlerle, bibiriyle yarışan alternatif çözümler arasından en iyi çözümü seçer.

Bu tezde, önce genetik algoritmanın genel tanımı ve genetik algoritmanın verimliliğinin sebeplerini açıklayan teorik altyapısı, Şema teoremi verilir. Daha sonra, Bahçeşehir Üniversitesi Fen Edebiyat Fakültesi verileriyle gerçek bir üniversite ders programlama problemi denenir. Bu probleme genetik algoritma tabanlı bir çözüm önerilir. Genetik algoritma uygulamaları anlatılır ve örnek probleme uygulanır. Ayrıca, popülasyon büyüklüğü, çaprazlama ve mutasyon oranları gibi parametrelerin

en iyi değerlerinin tespiti denenir. Bu problemde sonuç olarak sadece kabil ders programları üreten, modifiye edilmiş, kati kısıtların ihlaline izin vermeyen genetik operatörlerle çalıştık. Bu yüzden amaç sadece iyi kalitede ders programı üretmekti. Bu bize daha kısa sürede daha iyi sonuçlar elde etmeyi sağladı.

Anahtar Kelimeler: Sezgisel Yöntemler, Zaman Tablolama, Genetik Algoritma, Kısıtlar

CONTENTS

TABLES

FIGURES

1. INTRODUCTION

Timetabling is defined as “the allocation, subject to constraints, of given resources to objects being placed in space-time, in such a way as to satisfy as nearly as possible a set of desirable objectives “(Wren 1996, p53). A more common definition can be given as assigning a set of events (lectures, vehicles, public events, etc.) to limited set of resources over time in such a way to satisfy predefined constraints. These constraints can be categorized as hard constraints and soft constraints where hard constraints have higher priority to be satisfied than soft constraints.

There are various timetabling problems in real world such as allocation of events, activities, people, vehicles, etc. In most of the cases constructing a workable timetable is very difficult to achieve because the resources (time, place, people, etc.) are limited. So constructing an efficient timetable becomes a crucial problem for people.

This study focuses on the timetabling problem that every educational institution faces with at the beginning of each academic semester. It is the problem of allocation of events (teaching sessions, exams, lab sessions, etc) within a given number of rooms and time periods.

The problem of educational timetabling can be arranged into two categories as course timetabling and exam timetabling. It is clear that there are a lot of similarities between these two. The main difference is that in exam timetabling problem, although it is not preferable, two or more exams can be scheduled in the same room, at the same time. However in the course timetabling problem, two different courses are not allowed to be scheduled in the same room, at the same time. Another crucial difference is flexibility of examination’s time. An exam can be scheduled at weekends or in the evenings while courses cannot. Moreover in exam timetabling problem conflicts are strict. Every student must take his/her examinations at its time. In addition to these, in exam timetabling each student should take at most one exam each day and also too many consecutive exams are not allowed.

Course timetabling problem can also be divided into two: University timetabling and school timetabling. School timetables are more compact, meaning that there should be

no free periods between lessons. Students have same starting, finishing and lunch time which is not the case for university timetables. There are more programs, classes, lecturers and different group of students in university timetabling problem which makes the problem more complex.

Timetabling problems has been proved to be in the class of NP-hard problems (Bardadym 1996). Here NP stands for "non-polynomial". What NP-hard means is that the amounts of work needed to solve this type of problems are increasing exponentially with the problem size. Therefore they cannot be solved in polynomial time.

Solutions to timetabling problems have been proposed since the 1960s. Many procedures are used for constructing workable and attractive timetables. These approaches can be grouped into three categories: artificial intelligence, human-machine interaction and operations research (Lai *et al*. 2006). Details of different approaches which are tried to solve the timetabling problem are given at literature review part. This research is focused on solving university course timetabling problem using Genetic Algorithms which is categorized as metaheuristic methods in operations research subset.

To give a brief explanation, genetic algorithms are methods which use algorithms inspired by the processes of neo-Darwinian evolutionary theory. In a genetic algorithm each 'chromosome' represents a candidate solution to the problem. By evaluating and ordering the performance of a set of candidate solutions, 'parents' of new candidate solutions ('children') are chosen. Applying 'mutation' and 'crossover' operations to the children gains the population new candidate solutions with higher performance to get a better solution. These new set of candidates then evaluated, and then this cycle continues until a termination criteria is occurred. A termination criterion can be a predefined number of cycles or finding an adequate solution. More detailed information will be given at Chapter 3.

The aim of this research is to describe and experimentally verify the Genetic Algorithm which is applicable to course timetabling problems. Moreover, with such an algorithm, we would like to tackle a real-life course timetabling problem for Bahcesehir University. This work was motivated to solve the timetabling problem and produce a solution fully acceptable by all users.

This thesis is organized as follows. Chapter 2 is literature review. Here various approaches to solving the university timetabling problem are given. Chapter 3 provides an overview of Genetic Algorithms. Here fundamentals of genetic algorithm and its theoretical aspects are told. Moreover a simple example demonstrating how genetic algorithms work is given. Chapter 4 is the problem definition part. It defines the university course timetabling problem with all its goals, constraints and data acquisition. Chapter 5 describes the implementation of Genetic Algorithm to the data coming from Bahcesehir University. Chapter 6 presents the results of the sample case study. Chapter 7 discusses the results of the implementation and finally concludes the thesis.

2. LITERATURE REVIEW

As stated earlier in this study, timetabling problems are in the set of NP-hard problems. It means that, for scheduling C courses into T time slots, there are T^C possible candidate timetables. Clearly, using the manual way to generate workable timetables is not preferable in such a case. Generating timetables through automatic methods then seems to be an attractive alternative to manual approach. This chapter concentrates on the evaluation of automated timetabling.

Much of the literature on timetabling problems dates from the 1960s when first computers could be useful in timetabling. A wide variety of methods used to solve timetabling problems. There is not a strict discrimination to categorize these methods. Some resources classify a method into some group while other resources study that method in another group. This chapter reviews the literature on timetabling on the basis of the classification of Lai *et al*. (2006). They first classified the approaches to timetabling problem into three categories: artificial intelligence, human-machine interaction and operations research. Constraint programming (Deris *et al*.2000, Valouxis and Housos 2003), Expert Systems (Solotorevsky *et al*. 1994, Gunsdhi *et al*. 1996, Isaai and Cassaigne 2001) and Neural Networks(Smith et al. 1995, Carrasco and Pato 2001) can be classified in artificial intelligence technologies. The human-machine interaction approaches are iterative processes. First the machine finds a solution then the user either accepts it or improves it through instructions. This iteration goes on until the user is satisfied or additional improvement is not possible. Human-machine interaction methods are used by Mulvey (1982) and Johnson (1993) for timetable construction. Operations research approaches also have been grouped into three categories: sequential methods, clustering methods and metaheuristic methods. This research wills attempt to solve the timetabling problem by Genetic Algorithm that can be categorized into operations research group and metaheuristics subgroup.

2.1 OPERATIONS RESEARCH APPROACHES TO TIMETABLING PROBLEMS

2.1.1 Sequential methods

In these methods, courses are first ordered according to the difficulty level of scheduling them and then assigned sequentially into valid time periods so that no courses in the period are in conflict with each other (Carter 1986). Graph coloring algorithm is the most common example of sequential methods.

The graph coloring problem (GCP) is defined as to color the vertices of a graph G=(V,E), where $V=\{v_1,v_2,...,v_n\}$ is the set of vertices and E the set of edges connecting the vertices such that no pair of vertices with a common edge is not in same color and also the number of colors used is minimum.

Using graph coloring to solve small timetabling problems has a long history (Broder 1964, Welsh and Powell 1967, Krarup and de Werra 1982, Asratian and de Werra 2002). Simply, a timetabling problem can be converted to a graph coloring problem by constructing a graph considering each course/exam as a vertex, and considering the edges between any pair of vertices as pair of courses/exams that cannot be scheduled to the same timeslot. Each color represents a timeslot. If one can manage to color the graph with no more colors than available timeslots, a solution is found. A simple example of finding a solution to the timetabling problem with 10 courses/exams and 5 available timeslots is given by Lewis (2007).

However, it is noteworthy to indicate that, conversions to graph coloring problems adapt well when dealing with small-scale timetabling problems. When real world timetabling problems are considered, graph coloring method fails to be sufficient (Tripathy 1984).

2.1.2 Clustering methods

In these methods, first the courses which are satisfying hard constraints are split into groups and then the groups are assigned into timeslots using various optimization techniques to fulfill the soft constraints. One of the first papers describing this method is written by White and Chan (1979). The main disadvantage of these methods is that the

clusters of courses are formed and fixed at the beginning of the algorithm. This may result a timetable with poor quality.

2.1.3 Metaheuristic methods

Metaheuristic approaches have been widely developed and applied to university timetabling problems over the last two decades or so. Metaheuristic methods begin with some initial solutions and try to get better solutions by using search techniques. Other search techniques like local search, greedy algorithm are stuck in local optima and not able to find the global optima. Metaheuristic methods like simulated annealing, tabu search, ant colony systems and genetic algorithms can cope with this problem although they cannot guarantee to find the best solution because of the nature of the problem type. These methods are reviewed as metaheuristic approaches in this study.

Simulated annealing is said to be the oldest among the metaheuristic methods. It is an optimization method that inspires from physical cooling process of solids. The algorithm starts with an initial state which is selected randomly in search space as initial solution and a high initial temperature. As in all metaheuristic approaches, the aim in simulated annealing is improving the initial solution. Although the new solutions improving the initial solution are always accepted, the non-improving solutions are accepted with a certain probability. This probability of accepting a non-improving solution is calculated according to the current temperature of the system. So algorithm begins with a high probability of accepting non-improving solutions corresponding to high initial temperature and as the solids cools gradually, the temperature of the solid decreases therefore the probability of accepting non-improving solutions decreases. At low temperatures the algorithm accepts only improving solutions. It goes on until a stopping criterion is occurred or a satisfactory solution is reached.

Step 1. Generate initial solution x.
Step 2. Assign Temperature to Initial Temperature.
Step 3. Generate candidate solution x' from current solution x.
Step 4. If fitness (x') > fitness(x) then $x = x'$.
Step 5. If fitness (x') ≤ fitness(x) then calculate Acceptance Probability.
    5.1 If Acceptance Probability > random [0,1] then $x = x'$.
Step 6. Update Temperature according to cooling schedule.
Step 7. If termination condition is met finish, otherwise go to Step 3.

Figure 2.1: Pseudocode for Simulated Annealing.

The analogy between minimizing the cost function of combinatorial optimization problems and annealing is identified by Kirkpatrick *et al*. (1983). Afterwards, simulated annealing has been used to solve many types of combinatorial optimization problems including timetabling problems (Abramson 1991, Thompson and Dowsland 1998, Azimi 2005) because it is easy to implement. Main advantage of simulated annealing is its ability of escaping local optima by exploring other areas of the solution space. On the other hand, long running time is the disadvantage of this method.

Tabu Search is another metaheuristic method. It starts from a random initial solution and successively moves to one of the neighbors of the current solution. The difference of tabu search from other metaheuristic approaches is based on the notion of tabu list, which is a special short term memory. That is composed of previously visited solutions that include prohibited moves. In fact, short term memory stores only some of the attributes of solutions instead of whole solution. So it gives no permission to revisite solutions and then avoids cycling and being stuck in local optima. During the local search only those moves that are not tabu will be examined if the tabu move does not satisfy the predefined *aspiration criteria*. These *aspiration criteria* are used because the attributes in the tabu list may also be shared by unvisited good quality solutions. A common *aspiration criterion* is better fitness, i.e. the tabu status of a move in the tabu list is overridden if the move produces a better solution.

Step 1. Generate initial solution x.
Step 2. Initialize the *Tabu List*.
Step 3. While set of candidate solutions X' is not complete.
  Step 3.1 Generate candidate solution x' from current solution x
  Step 3.2 Add x' to X' only if x' is not tabu or if at least one *Aspiration Criterion* is satisfied.
Step 4. Select the best candidate solution x^* in X'.
Step 5. If fitness(x^*) > fitness(x) then $x = x^*$.
Step 6. Update *Tabu List* and *Aspiration Criteria*
Step 7. If termination condition met finish, otherwise go to Step 3.

Figure 2.2 : Pseudocode for Tabu Search

A short general description of tabu search and its applications is given by Glover and Laguna (1997). Hertz (1991) and Alvarez-valdes *et al*. (2002) use tabu search technique to solve the university course timetabling problems.

Ant colony systems are metaheuristic approaches introduced by Dorigo *et al*. (1991) and inspired from behavior of ants finding the shortest path from their nest to food sources. Ants can manage this by producing a special liquid called *pheromone*. They deposit it on the ground while they are walking. The ants, which choose the shortest path by chance at the beginning, arrive the nest quickly after visiting the food source. While the ants following the longer path are on the way of return, the quicker ants leave the nest for food second time. So the quantity of pheromone deposited on the shortest path is more than the longer ones. The pheromone on the longer paths starts to evaporate in time because they are not used. When an ant decides which path to follow, it most probably prefers the one with the higher amount of pheromone. Hence the shortest way will be chosen by the great majority of the ants in time.

Step 1. Initialize pheromone values
Step 2. Release each ant in the colony to construct an independent solution through components
Step 3.Update pheromone values
Step 4. If termination condition is met finish, otherwise go to Step 2

Figure 2.3 : Pseudocode for Ant Colony System

In Figure 2.3 pseudocode for ant colony system is given. The algorithm starts with the initialization of pheromone values. At each iteration of the algorithm, each ant is released to construct a new solution independently from other ants. Construction of the solution is performed by adding solution components to the partial solution constructed so far. These solutions are used to update the pheromone values. The algorithm continues until a stopping condition is met.

Socha and Samples (2003) suggested how to implement ant colony systems to university timetabling problems. At each step, each of the ants constructs a complete timetable using heuristics and pheromone information. Timetables are then improved using a local-search procedure, and results are written back to the pheromone matrix to be used in the next iteration. And Azimi (2005) implements ant colony systems to the exam timetabling problem. He compares ant colony systems with tabu search, simulated annealing and genetic algorithms. He declares that the ant colony systems give the best solution.

Genetic algorithms are metaheuristic methods that try to find solutions to NP-hard problems through evolution. The main idea of genetic algorithms is to generate an initial population randomly and then evolve this population after a number of iterations. To do this, first the initial population is evaluated according to some criteria and then parents of the next generation are chosen with respect to the results of this evaluation. Then parents are mated to produce offsprings, namely the members of the next generation. This goes on until a stopping condition is met.

Mutation and crossover are two crucial operators of genetic algorithm. Mutation refers for small variations in the genetic material of the offspring and its purpose is to diversify the next generation. On the other hand, crossover refers to recombination of two parents to produce offsprings and its purpose is to propagate good genetic material from parents to offspring.

Step 1. Generate initial population.
Step 2. Evaluate population.
Step 3. Apply Crossover to create offspring.
Step 4. Apply Mutation to offspring.
Step 5. Select parents and offspring to form the new population for the next generation.
Step 6. If termination condition is met finish, otherwise go to Step 2.

Figure 2.4 : Pseudocode for Genetic Algorithm

Genetic algorithms were first suggested by Holland (1975) in his of book *Adaptation in Natural and Artificial Systems.* Colorni *et al*. (1990) constructed the first successful timetable using genetic algorithm. Several researchers have used genetic algorithms to solve timetabling problems during the last few decades (Ergul 1996, Gen and Cheng 1997, Syarif *et al*. 2002, Aytug *et al.* 2003, Chaudhry and Luo 2005). Irene *et al.* (2007) reviews the literature on approaches used in the timetabling problem and presents a comparison of 19 papers in their study.

Genetic algorithms have several advantages when compared to other optimization techniques. First of all, genetic algorithms do multiple directional searches with a set of candidate solutions while other methods perform single directional searches. Moreover while other methods deal with decisions variables, genetic algorithms represent solutions in terms of coding. Domain knowledge is used only to code the problem and for establishing fitness function.

3. GENETIC ALGORITHMS: AN OVERVIEW

Genetic Algorithm is an optimization method that is mainly based on Darwin's theory of evolution and current knowledge of genetics. As in nature, genetic algorithms include concepts such as chromosomes, genes, mating, crossover, mutation, and evolution, too. So it is useful to remind some of this biological terminology that will be used.

Biological Terms	Genetic Algorithm
Chromosome	Solution (coding)
Gens	Part of solution
Alleles	Values of gene
Phenotype	Decoded solution
Genotype	Encoded solution

Figure 3.1 : Explanation of biological terms in Genetic Algorithm

All living organisms consist of cells which include chromosomes. A chromosome is formed by millions of genes each of which encodes a specific characteristic of the organism, such as blood type or eye colour in humans. Such characteristics are called *phenotypes*, for example, *0, A, B*, *AB* for blood type or hazel, blue, brown for eye colour. The particular settings of genes are called *genotype* and values of the genes are called the *alleles*. An organism's phenotype is formed by its genotype.

In nature, most species are *diploid* meaning that their chromosomes are arrayed in pairs. Human beings are also diploid and they have 23 pairs of chromosomes in each body cell. During *crossover*, genes in each parent are exchanged between each pair of chromosomes to form a single chromosome, namely a *gamete*. Then one gamete from mother and one from father pair up to create a full set of diploid chromosomes for child. This explains how certain genotypes are inherited from a person to his/her child. Occasionally, some gene values change because of external effects. This kind of change in the gene value is called *mutation*.

Individuals who are more adapted to the environmental conditions have higher probabilities for survival, therefore they have higher probabilities for producing their children. This also means that, the individuals and their genes which are better adapted most likely to tend to remain while those that are not adapted tend to become extinct as time passes. This natural process of survival of the fittest is called Darwinian evolution.

Over the last two decades, genetic algorithms have been extensively used as search and optimization methods in various problem domains such as sciences, engineering and management. Although they do not guarantee to find optimum solutions, they are successful in finding solutions with high acceptance. Their broad applicability and ease of use appeals the researchers to choose genetic algorithms. The term chromosome typically refers to a candidate solution to the given problem in genetic algorithms. By analogy with biology, the chromosome is referred to as the genotype, the encoded solution, whereas the solution it represents is known as the phenotype, the decoded solution. Crossover consists of exchanging genetic material between two single chromosomes of parents. Mutation consists of replacing a randomly chosen gene with a randomly chosen new gene. In this chapter, the working principle of genetic algorithms is described and also a simple example of genetic algorithms is given.

3.1 FUNDAMENTALS OF GENETIC ALGORITHMS

3.1.1 Chromosome representation

Designing chromosome structure is very crucial in genetic algorithms, because each chromosome embodies a solution and its representation affects the performance of algorithm. At the beginning of the genetic algorithm, the researcher decides how to design the chromosomes according to the problem type and which solution form will be adequate for the algorithm. There are two approaches in designing chromosome structure. First one of them is direct encoding which directly represents a solution by a chromosome. However, direct encoding is not always possible or efficient for each problem. In such a case, indirect encoding is used and now each chromosome is encode only the instructions of building a solution instead of the solution itself.

Most conventional representation of a chromosome in direct encoding is a bit-string of characters in which each chromosome consists of a string of genes whose allele values are characters from the alphabet {0, 1}. And the length of the string differs according to the problem nature and the information that one wants to encode. For example, a bit string of length 10 might be used to encode a single integer value in binary notation in one problem, on the other hand, the bits might encode the presence or absence of 10 different factors (if it is present then code 1, if it is absent then code 0) in another.

However genes can be represented in the form of integers, reals, arrays, trees, matrix, graphs or any data type. In course timetabling problems, a chromosome is encoded as weekly timetable of set of courses and generally in the form of a matrix.

3.1.2 Fitness of a solution

Fitness function is a measure that evaluates the quality of the chromosome as a solution to a particular problem. The fitness function is problem specific. If the aim is minimizing a function, for instance think of a company tries to minimize its cost, then the fitness function is the cost itself. In most of the timetabling problems, quality of a chromosome is measured by number of constraints satisfied.

3.1.3 Basic steps of genetic algorithm

3.1.3.1 Initialization of the population

A set of candidate solutions are called population. At the beginning of the algorithm, to initialize the population, a predefined number of solutions are generated randomly.

3.1.3.2 Selection

After the initial population is constructed, it is time to the process of choosing parents (chromosomes) for breeding, namely selection of parents. In genetic algorithm, selection is designed to use fitness function as a discriminator of the quality of solutions represented by the chromosomes in a population. Those with higher fitness value have a greater chance of being selected than those with lower fitness value.

There are various selection methods in genetic algorithm. The standard selection method used is Roulette Wheel or in other words, Fitness Proportional selection. In this kind of selection method, expected number of times that a chromosome will be selected for breeding is calculated as chromosome's fitness divided by the average fitness of the population. To implement this method, each chromosome is placed as a slice of a circular roulette wheel, the size of the slice is proportional to the chromosome's fitness. If there are N chromosomes in the population, the wheel is rotated N times. On each rotation, the chromosome shown by the wheel's marker is selected to be in the pool of parents for the next generation. So the selection is end up with N chromosomes, keeping initial number of chromosomes in the population. Higher fitness value of chromosome

implies the larger slice on the roulette wheel, therefore the probability of being selected is higher. Note that one chromosome can be chosen more than once.

Tournament Selection is another method for deciding which chromosome will be chosen. In this method, two chromosomes are chosen randomly in the population and then the one with the highest fitness between these two is selected as the parent. The two chromosomes are the return the original population and can be selected again.

3.1.3.3 Recombination

After selection, chosen chromosomes are recombined to create new members of next generation. New members are expected to be better than their predecessors since selection for recombination gives more chance to the ones with higher fitness to be chosen. There are two main recombination operators in genetic algorithm: crossover and mutation.

3.1.3.3.1 Crossover

The idea of the crossover operation is mixing genetic material from two selected parent chromosomes to produce child chromosomes. If chromosomes are represented in bit-strings, this is done as follows. First of all a crossover probability is defined. A random number in the interval [0, 1] is generated and compared with the crossover probability. If the crossover probability is greater than or equal to the random number then the crossover operator is applied. If the random number is greater than the crossover probability then crossover operator is not applied. Therefore both parents remain unchanged, so the children chromosomes are exact copies of their parents. The value of the crossover probability has great importance here. It can either be defined experimentally or can be defined by schema theorem principles (Goldberg 1989, 2002; Goldberg and Sastry, 2001) which will be explained later in this chapter.

There are many methods used in designing crossover operator in literature. Although they can be changed by problem type, the most common used crossover operators are as follows:

***N*-point Crossover**: One-point and two-point crossovers are the simplest and widely used crossover methods. In one-point crossover, a number less than the length of

chromosome is selected randomly and represents crossover point. Then, the genetic material before the crossover point remains unchanged while the genetic material after the crossover point is exchanged between the parents. The idea of one-point crossover can be generalized to *N*-point crossover by using *N* crossover points rather than just one. For instance, in two-point crossover, two crossover points are selected randomly and the genetic material between crossover points is swapped. An example of two-point crossover method is given in Figure 3.2.

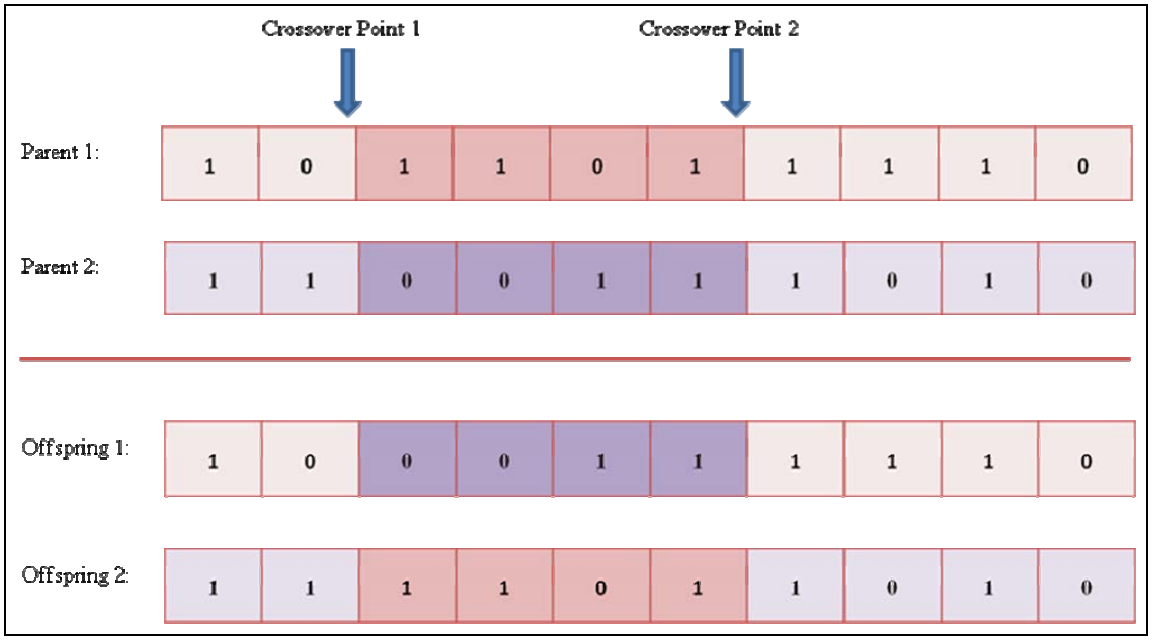

Figure 3.2: Two-point Crossover

Uniform Crossover: In uniform crossover each allele has the chance of exchanging with the gene of the other parent with the probability of predefined *swapping probability*. Most of the time 0.5 is taken as swapping probability value.

Partially Matched Crossover (PMX): In partially matched crossover, two parents are randomly selected and two crossover points are generated randomly. Alleles within the two crossover points of a parent are exchanged with the alleles corresponding to those mapped by the other parent. Figure 3.3 illustrates an example of partially matched crossover. According to the example, firstly the genes between two crossover points are swapped between two parents. Then the first gene value in Parent 1 within the two

crossover points, 7, maps to 9 in Parent 2. Therefore, genes 7 and 9 are swapped in Parent 1. Similarly 8 and 2, also 4 and 1 are exchanged to create the offspring Offspring 1. Corresponding changes are done in Parent 2 to create the offspring Offspring 2.

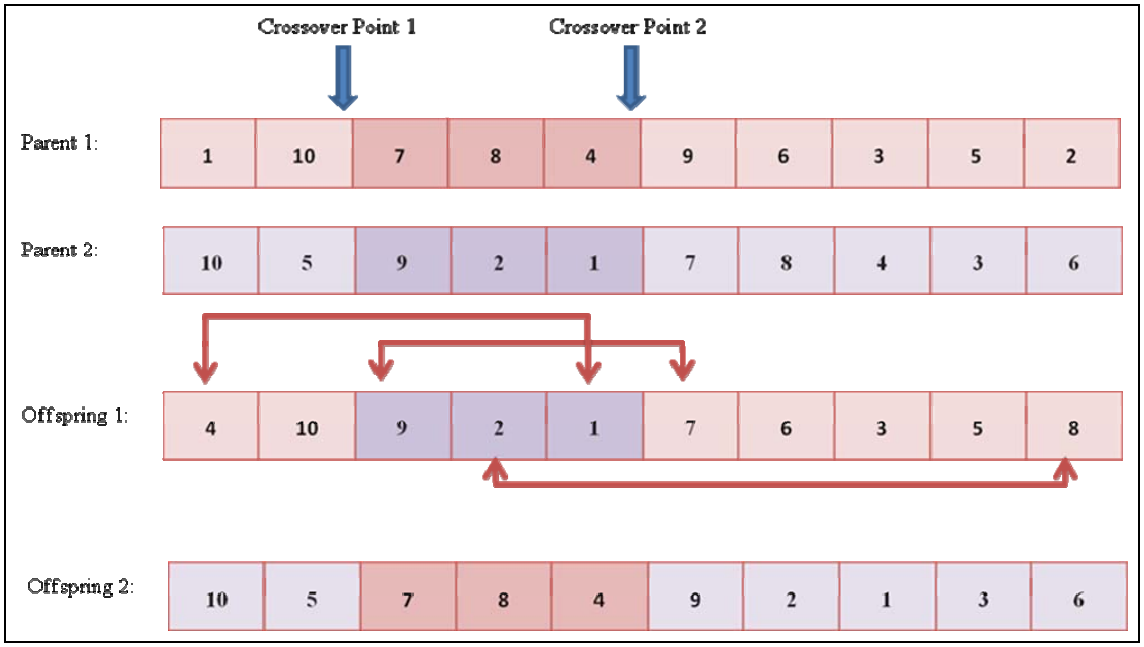

Figure 3.3: Partially Matched Crossover

3.1.3.3.2 Mutation

In genetic algorithms, most chromosomes in the population seem like each other after a number of generations. What this means is that, no crucial changes in the population, therefore in the search space do not occur. To overcome this problem and add diversity to the population mutation operator is used. Mutation operator has the effect of creating a new chromosome which cannot be created by the ordinary crossover operator.

After crossover is applied and the offsprings are formed, they have a chance of being mutated. Mutation operators changes one or more gene values in a single chromosome. For the chromosomes represented in binary strings, the mutation operator works as follows. A number in the interval [0, 1] is generated randomly and compared to a predetermined mutation rate. Mutation rates used in literature are usually very small (e.g. 0.001). If the random number is greater than the mutation rate, mutation is not applied. If the mutation rate is greater than or equal to the random number, then the gene value is changed artificially from 0 to 1 or 1 to 0. (See Figure 3.4)

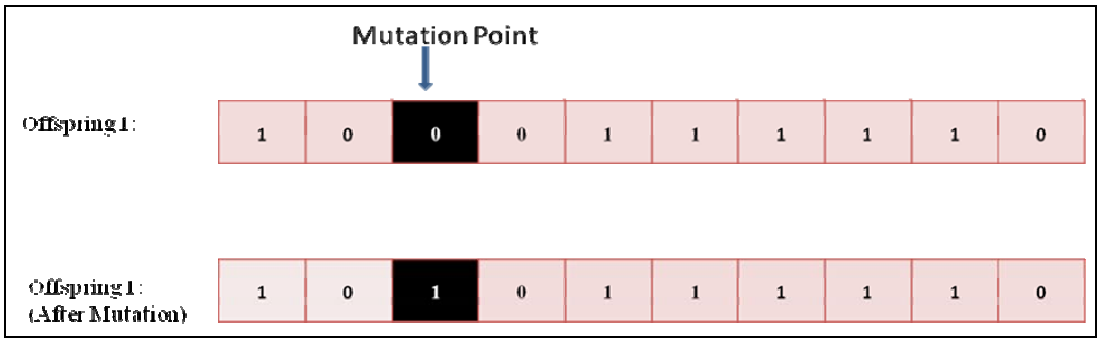

Figure 3.4: Mutation Operator

3.1.3.4 Replacement

After the new offsprings are created with crossover and mutation operators, it is time to form up the successor generation. Recall that parent chromosomes were selected according to their fitness, so it is expected that the offsprings increase the fitness of the population generation by generation. Through replacement, genetic algorithm decides whether offspring will survive or will become exist. Some of the most common replacement techniques are explained below.

Complete Replacement

This technique deletes all the members of the predecessor population and replaces them with the same number of new chromosomes that have just been created.

Steady-state

In this technique, n old members are chosen in the population and replaced with n new members. The choice of the number n and the decision of which members to delete from the current population are important aspects of genetic algorithm.

Replacement with elitism

This technique is same as complete replacement except that this time one or two chromosomes with the highest fitness are chosen to next generation. By this way, good solutions are preventing from become extinct.

3.2 A SIMPLE EXAMPLE OF GENETIC ALGORITHM

The aim of this section is to show how basic components of genetic algorithm work through optimization of a function. The function is defined as

$$f(x) = x\sin(\frac{1}{x})$$

and drawn in Figure 3.5. The problem is to find a minimum for this function on the interval [0, 0.5], i.e. to find a x_0 such that $f(x_0) \leq f(x) \quad \forall x \in [0, 0.5]$. Actually, maximum or minimum values of a function can be calculated by the zeros of the first

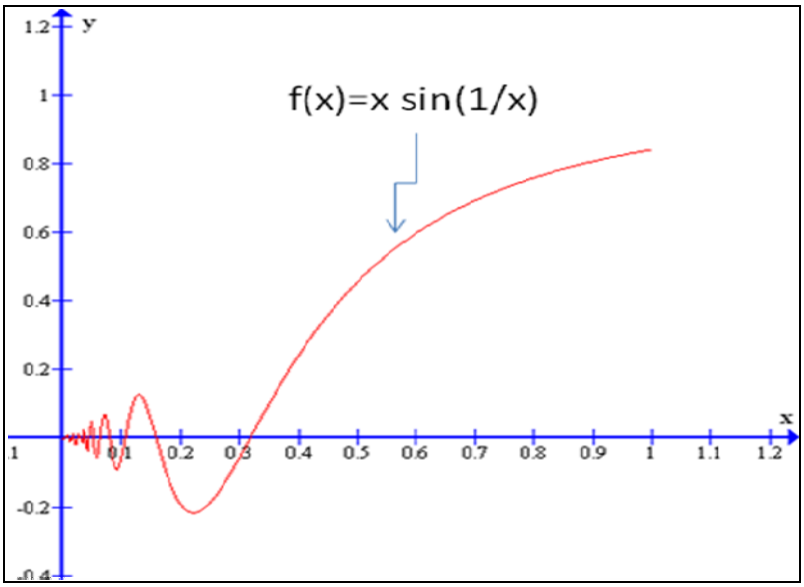

Figure 3.5: Graph of the function f(x) = xsin (1/x)

derivative where $f'(x) = \sin\left(\frac{1}{x}\right) - \frac{\cos\left(\frac{1}{x}\right)}{x} = 0$. The formula equals to $x\tan\left(\frac{1}{x}\right) = 1$ and has infinitely many solutions. However we wish to construct a genetic algorithm to solve the above problem.

Representation:

Binary vector is used to represent real values of x as a chromosome. A binary vector consists of 32-bits since the processor allows this number. This means that the length of the domain of variable x, which is 0.5 in our example, is divided 2^{32}-1 equal size ranges. So converting a binary string to a real number is completed as follows. First binary string of length 32 is converted from base 2 to base 10:

$$x^* = ((b_{31}b_{30....}b_0))_2 = \left(\sum_{i=0}^{31} b_i 2^i \right)_{10}$$

Then a corresponding real number is found by:

$$x = x^* \frac{1}{2^{32}-1} 0.5 \text{ where 0.5 is the length of the domain.}$$

For example, a chromosome (11110011110010101001011110100111) represents the number 0,476155032 in [0, 0.5] since

$$x^* = (11110011110010101001011110100111)_2 = (4090140583)_{10}$$

and the real number is

$$x = (4090140583) \frac{1}{2^{32}-1} 0.5 = 0{,}476155032.$$

One can easily see that (00000000000000000000000000000000) and (11111111111111111111111111111111) represent the boundaries of the domain, 0 and 0.5, respectively.

Initial population:

We construct randomly 10 chromosomes of each is a binary vector of 32 bits as initial population. The 10 chromosomes c_i where $1 \le i \le 10$ and their corresponding real value x_i on the interval [0, 0.5] is listed below.

c_1 = (01101110010010110001111111101001) x_1 = 0,21541691

c_2 = (11000000001100100111010011011111) x_2 = 0,37538495

c_3 = (11110011110010101001011110100111) x_3 = 0,47615503

c_4 = (10001100010110101011110101110100) x_4 = 0,27412979

c_5 = (10010100101100110001000010100000) x_5 = 0,29042866

c_6 = (10000110110110000100100000000000) x_6 = 0,26336884

c_7 = (01100101000110110101101011110000) x_7 = 0,19747433

c_8 = (11010110010111110111001010110111) x_8 = 0,41869696

c_9 = (01011001000010001110001110010111) x_9 = 0,17389594

c_{10} = (11001010100101001111100010101010) x_{10} = 0,39566781

Evaluation function:

The evaluation function *eval* for chromosomes is equivalent to the original function *f*, namely *eval(c) = f(x)* where *c* represents the chromosomes and *x* represents the corresponding real value. Evaluation function plays an important role because it is used to rate the potential solutions in terms of fitness. For instance, the fitness of the chromosomes in our population is as follows:

$$\textbf{\textit{eval}}\ (c_1) = f(x_1) = \textbf{-0,214885912}$$

$$eval\ (c_2) = f(x_2) = 0{,}172565550$$

$$eval\ (c_3) = f(x_3) = 0{,}410983902$$

$$eval\ (c_4) = f(x_4) = -0{,}132941257$$

$$eval\ (c_5) = f(x_5) = -0{,}086269622$$

$$eval\ (c_6) = f(x_6) = -0{,}160509512$$

$$eval\ (c_7) = f(x_7) = -0{,}185396112$$

$$eval\ (c_8) = f(x_8) = 0{,}286388252$$

$$eval\ (c_9) = f(x_9) = -0{,}088302975$$

$$eval\ (c_{10}) = f(x_{10}) = 0{,}228031778$$

Clearly, the chromosome c_1 is the fittest chromosome in the population since its evaluation returns the smallest value. Moreover c_3 is the worst chromosome with the highest fitness value.

Genetic operators:

In genetic algorithm there is two classical operators: crossover and mutation. To apply one point crossover the first two fittest chromosomes c_1 and c_7 are chosen. Also a random number, say 21 is selected for crossover point. That is, the chromosomes are split at the 21st gene, and then the parts are exchanged between the chromosomes as seen below:

$$c_1 = (\mathbf{01101110010010110011} \mid 11111101001)$$

$$c_7 = (01100101000110110101 \mid \mathbf{01011110000})$$

Then the resulting offsprings are:

$$o_1 = (\mathbf{0110111001001011000110101111000}0)$$

$$o_2 = (01100101000110110101111111101001)$$

The corresponding real values and the fitness of these new chromosomes are calculated below:

$$f(o_1) = f(0{,}21541676) = -0{,}214885813$$

$$f(o_2) = f(0{,}19747604) = -0{,}185400708$$

The first offspring o_1 has a better fitness value than its parents however the improvement can only be seen ninth decimal point.

Then mutation is applied to the offsprings to get better chromosomes. For mutation, 6 random genes are selected and then changed. The bit value '1' is flipped to '0' or vice versa. Assume that the alterations below are done:

$$o_1 = (01101110010010110001101011110000)$$

$$o_{1m} = (011\underline{\mathbf{100}}10\underline{\mathbf{1}}10010110001101011110\underline{\mathbf{11}}0)$$

$$o_2 = (01100101000110110101111111101001)$$

$$o_{2m} = (011\underline{\mathbf{110}}01\underline{\mathbf{1}}00110110101111111101\underline{\mathbf{11}}1)$$

The corresponding real values and the fitness of these new chromosomes are calculated below:

$$f(o_{1m}) = f(0{,}22420582) = -0{,}217113164$$

$$f(o_{2m}) = f(0{,}23751354) = -0{,}208197846$$

Note that, after mutation both mutated offsprings have better fitness value than not only from their parents but also the offsprings which are not mutated. Mutated offsprings act as parents for the next generation, thus each iterations starts with a better population. Here only one generation is illustrated and the solutions are promising. Genetic algorithm finds optimal or near-optimal solutions gradually by this way.

3.3 THEORETICAL ASPECTS OF GENETIC ALGORITHM

Although genetic algorithms are easy to describe and implement, understanding the way they work is quite difficult. Schema theorem, fundamental theorem of genetic algorithm, tries to explain how genetic algorithm can find optimal or near optimal solutions in a set of all possible solutions. In this section, schema theorem is briefly explained.

First of all, a schema (or schemata) in other words “similarity template” should be defined for a better understanding of schema theorem. A schema can be basically defined as a pattern of chromosomes. For instance, think of a genetic algorithm where each chromosome is made up a string of five bits. And let 01010 and 01101 are given two chromosomes. A common expression for these chromosomes is “the chromosomes with first two bits are 01”. Thus 01* * * is a template, namely a schema defining these chromosomes. Here asterisks represent “do not cares” meaning that they can be either 0 or 1. And 0’s and 1’s are called fixed bits. Therefore a schema is a pattern which uses the symbols {0, 1, *} to define a set of chromosomes. One can easily see that a schema represents 2^n chromosomes in the population where n represents the number of asterisks in the schema. In our example, 01* * * represents eight chromosomes since there are three asterisks in the schema: {01000, 01100, 01110, 01111, 011101, 01010, 01011, 01001}. Obviously * * * * * represents whole population.

For a particular schema H, the *length* l_H is defined as the difference of positions of the first and last fixed bits (namely 0's and 1's) of H. For instance, for the schema 01* * *10* * *, the first fixed bit (0) is in position 1 and the last fixed bit (0) is in position 7. So the schema H has length 7 − 1 = 6. And the *order* of H is the number of fixed positions and is shown as o(H). Since 01* * *10* * * has 4 fixed bits, it has order 4.

Schema Theorem

Let *H* be a schema and let *mH(t)* be the number of chromosomes correspond to *H* present in population at time t of an evolving Genetic Algorithm. Then the expectation of the number of chromosomes correspond to *H* in population at time *t + 1*, denoted *mH(t+1)*, is given by the formula where *FH(t)* is the relative fitness of *H* which is defined to be the average fitness of all those chromosomes in the population belonging to H divided by the average fitness of all chromosomes in the population; p_c is the crossover probability; p_m is the mutation probability.

$$mH(t+1) \geq F_H(t)\, mH(t)\left[1 - p_c \frac{l_H}{l-1}\right]\left[(1-p_m)^{H(o)}\right]$$

Schema theorem explains which particular schema is likely to propagate through how the performance of genetic algorithm is affected by different choice of designs. For example, the theorem say that schemas with relative fitness *($F_H(t)$)* greater than 1are most likely to be represented in the next generation as a result of fitness proportional selection process. Moreover, schemas with a length l_H close to length of the chromosome have less chance to survive because of crossover. And also schemas with many defined bits are also more likely to become extinct as a result of mutation operator. Finally, one can conclude that short length, low order schemas having fitness above the average are most likely sampled in successor generations. Such schemas are called building blocks and can be defined as partial solutions. A chromosome which has many building blocks will be near-optimal solution.

4. PROBLEM DEFINITION

The university timetabling problem consists of a set of courses that are to be scheduled into a set of timeslots and a set of rooms and also consists of a set of constraints that are expected to be satisfied. Solving this problem is a complicated and long job and it needs to be identified in detail. In this section, all parts of a particular university course timetabling problem are defined step by step.

4.1 GOALS:

The main goal of creating a timetable is to provide faculty members and students with a schedule that is not only conflict-free but also has good quality. Different aspects of this goal in terms of faculty members, students, rooms are investigated as follows. From faculty members' point of view, the resulting timetable should be conflict-free at first. Ideally, it should leave at least one day free for academic purposes. From students' point of view, the timetable must be conflict-free for all groups of students. It should minimize idle hours between courses in a day, but it should give at least one hour break for lunch between 11:30 and 13:30. The consecutive courses should be in the nearby rooms. Finally, from rooms' point of view, the student group size should fit the room's capacity and two different courses should not be scheduled at the same time in a room. Moreover, each course should be scheduled in an appropriate room, such as a tutorial class or a lab.

4.2 CONSTRAINTS:

The goals described above are accomplished by means of predefined constraints. Thus, the timetabling problem aims to satisfy a set of constraints. These constraints are divided into two groups: hard constraints and soft constraints. Hard constraints are the strict ones therefore violation of them is not allowed. A timetable is called "feasible", meaning that it can be used by the university it was made for, when all hard constraints are satisfied. However, satisfaction of all soft constraints is not necessary for a timetable, but the "quality" of a timetable increases by the number of soft constraints satisfied. Thus the main aim in course timetabling is constructing feasible timetables with high quality.

Hard constraints:

Although the hard constraints used in academic institutions are problem specific, some hard constraints are common it most course timetabling problems. There is a list of widely used hard constraints (Pongcharoen *et al.* 2007).

H_1 : Students can only have one lecture at a time.

H_2 : A lecturer can only give one lecture at a time.

H_3 : A room can only be used for one lecture at a time.

H_4 : Lecturers' must be available at lecture times they are given.

H_5 : Specific room requirements (such as labs) are taken into consideration.

H_6 : The capacity of rooms must match with student size.

H_7 : Two or more complementary blocks of a course cannot be scheduled in the same day.

Soft constraints:

Likewise hard constraints, soft constraints are also change by the academic institutions' preferences. A list of soft constraints that should be satisfied in order the timetable to be considered of high quality is given below (Pongcharoen *et al.* 2007).

S_1 : Lecturers should have at least one day free for academic studies.

S_2 : The timetables for rooms should be as compact as possible.

S_3 : The number of free periods in students' timetables should be minimized.

S_4 : An hour lunch break should be scheduled between 12:00 and 15:00

S_5 : Students should have consecutive lectures in the same building.

Burke and Petrovic (2002) pointed out that it is almost impossible to construct timetables satisfying all individual preferences.

4.3 ACQUISITION OF DATA:

In order to start construction of a timetable, all the necessary data must be available. A general course timetabling problem requires all information about resources (including

the number of rooms, their size and type, availability of lecturers and the information about who taught what, the number of hours a lecturer taught in a week, etc.), educational program requirements (which group of student must take which courses). Some specific information such as lecturers' preferences (one free day, morning or evening lessons, etc.), academic institution's regulations (certain days can be allocated social activities).

There are different sources for getting data. While most of the researchers use the unpublished data of their own academic institution, Burke and Petrovic (2002) used the data that are available online.

5. METHODOLOGY

In order to solve the university course timetabling problem, an automated timetable construction based on genetic algorithm has been designed and implemented. To demonstrate our approach, a course timetabling problem in the Faculty of Arts and Sciences at Bahcesehir University is used. In this chapter, the details of the implementation of the proposed novel genetic algorithm for a real life course timetabling problem are described.

5.1 CHROMOSOME REPRESENTATION

The chromosome representation is one of the most significant parts in genetic algorithm, since every chromosome must represent all the information required for the construction of a whole timetable. Moreover chromosomes should be represented in such a way that is efficient for genetic operations such as crossover and mutation and fitness calculation.

DAYS / HOURS	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
08:30-09:30	1	10	19	28	37
09:30-10:30	2	11	20	29	38
10:30-11:30	3	12	21	30	39
11:30-12:30	4	13	22	31	40
12:30-13:30	5	14	23	32	41
13:30-14:30	6	15	24	33	42
14:30-15:30	7	16	25	34	43
15:30-16:30	8	17	26	35	44
16:30-17:30	9	18	27	36	45

Figure 5.1: The University Week

In this study we are dealing with direct encoding where each chromosome represents a candidate solution. There are 5 working days from Monday to Friday and 9 hours each day (from 08:30 - 17:30) so there are totally $5 \times 9 = 45$ timeslots in a week (See Figure 5.1). We need to construct a weekly timetable for each room. So we design an array by

adding up the 45 timeslots of each room end to end as shown in Figure 5.2 and we called it as Rooms Timetable. To clarify, we can say that first 45 elements of the array correspond to the first room; second 45 elements from 46 to 90 correspond to second room and so on. Thus, this array is of size $45 \times number\ of\ rooms$. And also we have another array called Course List. The elements of this array are all courses given in a particular semester. Moreover elements also keep the information about the lecturer and duration of the course. A chromosome is represented by one dimensional, $m \times 1$ array where m is the number of courses in the Course List. So each element of the chromosome has a relation with the same numbered element of the Course List. Each gene in a chromosome (each array element), say the *chrm[i]*, holds the index value of the room timetable element where the i^{th} course was scheduled to.

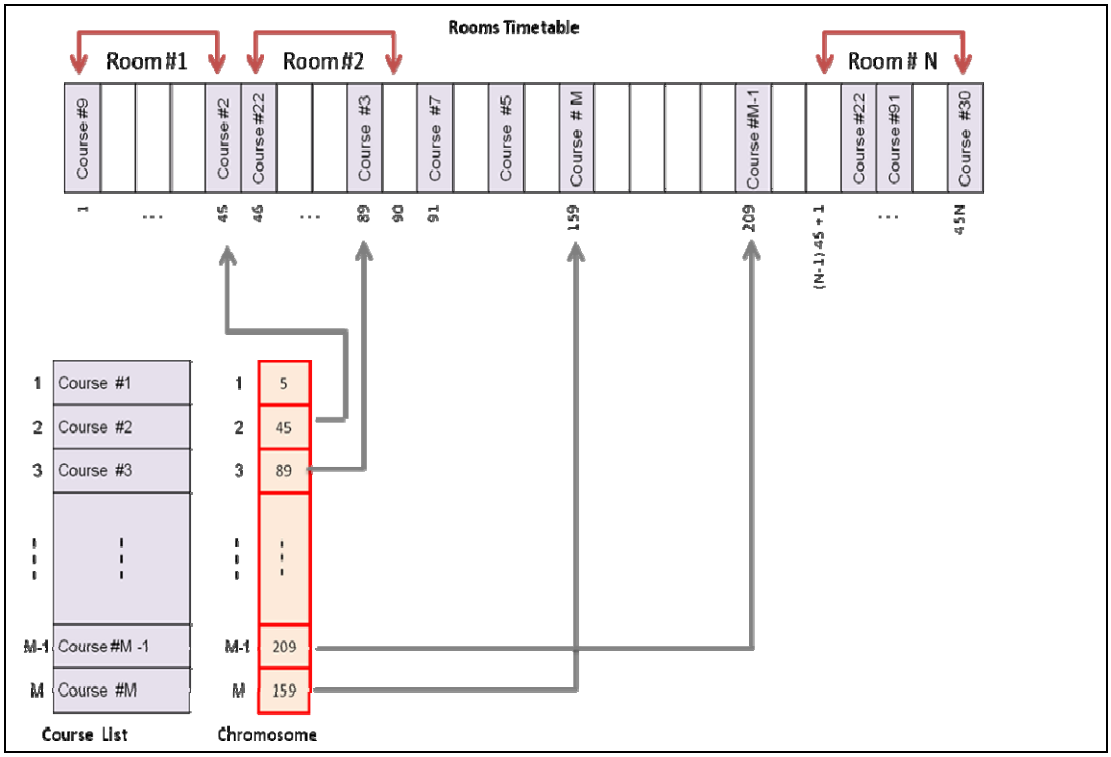

Figure 5.2: Chromosome Representation

For instance, the 2nd element of chromosome is 45, as seen in the Figure 5.2, since Course #2 was scheduled to the 45th element in the rooms' table. This means that Course #2 is given on Friday, starts at 16:30 at Room #1. Some courses are two or three hours long, so each chromosome element denotes the beginning time of each course.

We understand who gives that course and how long it takes from the information embedded in the Course List element.

5.2 FITNESS FUNCTION

Fitness function measures the goodness of the chromosomes. As the number of the constraints that a chromosome satisfied increases, so does the chromosome's goodness. So the evaluation of the chromosomes and the selection of the constraints are the other important steps of genetic algorithm.

In general, a fitness value for each chromosome is calculated with a reciprocal function of unsatisfied constraints. Some of most common used fitness functions are $\frac{1}{1+x}$, $\frac{1}{1+x^2}$ and $\frac{1}{1+\sqrt{x}}$ (Bhatt and Sahajpal, 2004). We used

$$1-\frac{\sum number\ of\ \ soft\ \ constra\,\text{int}\,s\ violated}{\sum number\ \ of\ \ soft\ constra\,\text{int}\,s}$$

in our study as fitness function. Thus, each chromosome has a fitness value between (0, 1] and if a chromosome satisfies all constraints it has 1 as fitness value.

Timetabling problem differs from other optimization problems by the presence of both hard and soft constraints. Hard constraints that must be satisfied in order to keep our timetable feasible are listed below:

- Lectures can be scheduled to only appropriate room (a lab or tutorial class)
- No lecturer or student group can have more than one class at a time.
- Room capacity and student group size must match.
- A classroom can only be used for one lecture at a time.
- Two or more complementary blocks of a course cannot be scheduled in the same day.

And also soft constraints that can be violated but they should be satisfied in order to get a timetable with high quality are listed below:

- Lecturers should have at least one day free for academic studies.

- The number of free periods between two consecutive lectures in students' timetables should be minimized.
- One hour lunch break should be scheduled between 12:00 and 15:00
- No student group should have less than 2, more than 7 hours lecture in a day.

Managing with both hard and soft constraints is an important aspect for timetabling problems. Lewis (2007) categorizes the metaheuristic algorithms into three groups according to their approach for dealing with the constraints.

One-Stage Optimization Algorithms: Satisfaction of both the hard and soft constraints is taken into consideration simultaneously. (Carrasco and Pato 2001)

Two-Stage Optimization Algorithms: Satisfaction of the soft constraints is only taken into consideration after a feasible timetable has been found. (Burke *et al.* 2003; Kostuch 2005)

Algorithms that allow Relaxations: Violations of the hard constraints are disallowed from the outset by relaxing some other feature of the problem. Attempts are then made to try and satisfy soft constraints, while also giving consideration to the task of eliminating these relaxations (Merlot *et al.* 2003).

In our study we construct a two-stage optimization algorithm. Unlike the classical genetic algorithms, violation of hard constraints is not allowed in any evolution step of chromosomes, i.e. all chromosomes represent a feasible solution. And then the violation of soft constraints are tried to be minimized. Thus, fitness function only considers only the satisfaction of soft constraints.

5.3 INITILIZATION OF THE POPULATION:

The initialization procedure is another important issue in all genetic algorithms because it should create a random initial population which spread in the whole search space. Diversity of initial population gives algorithm the opportunity to search the whole space of possible solutions and not to stuck with the local optima. In our study, 100 chromosomes are used for initial population and they are generated randomly.

5.4 SELECTION:

After the evaluation of the chromosomes, some of the chromosomes are chosen in order to create the next generation. We did the selection in two stages. At first, we rank the chromosomes according to their fitness value and then we chose first 20 chromosomes for breeding. At the second stage, we chose randomly 40 chromosomes other than first 20 chromosomes for breeding. The idea is preserving good genetic material by choosing the first 20 chromosomes, and adding diversity to the search by letting less fit chromosomes into the reproduction.

5.5 GENETIC OPERATORS:

Crossover:

In our study two-point crossover is used to mix the genetic material of two parents to produce offsprings with a crossover probability of p_c. The crossover procedure is described in the following:

1. Randomly choose crossover points.
2. Swap genetic material between two chromosomes.
3. In the case of duplicates, randomly choose a free and feasible timeslot and insert there.

Note that the important part of crossover procedure is coping with the duplicates. One or more genes in the changing part of the chromosomes can be seen twice in the resulting offspring. This means that two different courses start at the same time, at the same room. To overcome this conflict, we find a free timeslot and insert one of the conflicting courses there. Finally, we get a feasible offspring.

Mutation:

After we applied crossover to certain chromosomes, we also applied mutation with a low probability of mutation p_m and by the mutation procedure which is described in the following:

1. For each chromosome of which crossover applied before, run the following steps.

2. Generate a random number between 0 and 1.

3. If this random number is less than the mutation probability, then choose a gene randomly and change it by a feasible timeslot. Here feasible timeslot means a timeslot which is free in the timetables of lecturers, student groups and also rooms at the same time.

4. If this random number is greater than the mutation probability, then keep the chromosome un-mutated.

Both crossover and mutation is applied under the condition of satisfying hard constraints, i.e. the resulting chromosomes are also feasible.

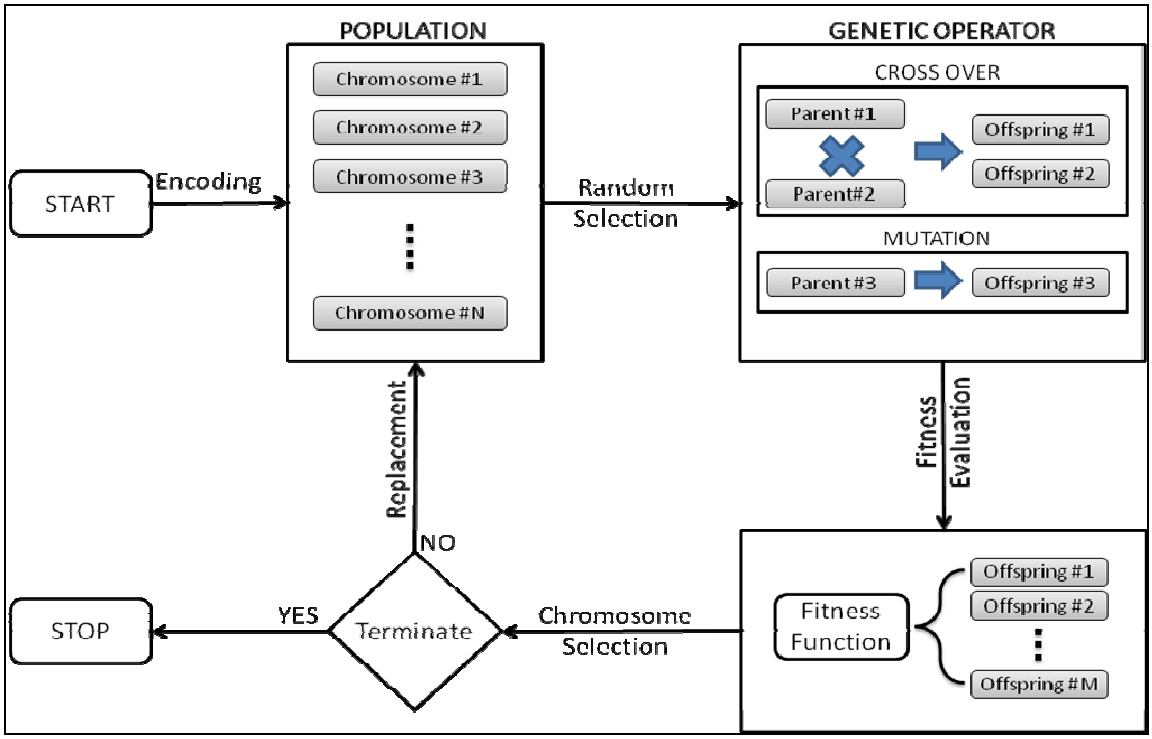

Figure 5.3: Flow chart of Genetic Algorithm

Replacement:

In order to keep the best chromosomes in every generation, we used a simple replacement strategy. The best 20 chromosomes of each generation are copied to the next generation. Then select 60 chromosomes randomly from remaining 80 chromosomes of the initial population and then replace with 60 offspring chromosomes

(20 of these 60 offsprings are produced from best 20 chromosomes of the initial population and remaining 40 offsprings are produced from randomly selected 40 chromosomes.). This replacement technique guarantees that the best chromosomes of each generation will be at least equal to the best chromosomes of the previous generation.

6. EXPERIMENTAL RESULTS

To demonstrate the efficiency and performance of the genetic algorithm we constructed, several experiments were carried out with the data coming from Bahcesehir University Faculty of Arts and Sciences. The faculty has 5 departments and more than 364 students. The model was tested with real data containing 114 courses to be scheduled into 45 timeslots and 12 rooms.

In the first experiment, we have aimed to analyze the convergence of the genetic algorithm. The genetic algorithm increases sharply when generation number is between zero and a thousand, however after 5000 generations nearly the same fitness value is observed (See Figure 6.1). Although we reached 30000th generation, we did not get a solution with a fitness value of 1. It is almost impossible that generating timetables which satisfy all individual preferences.

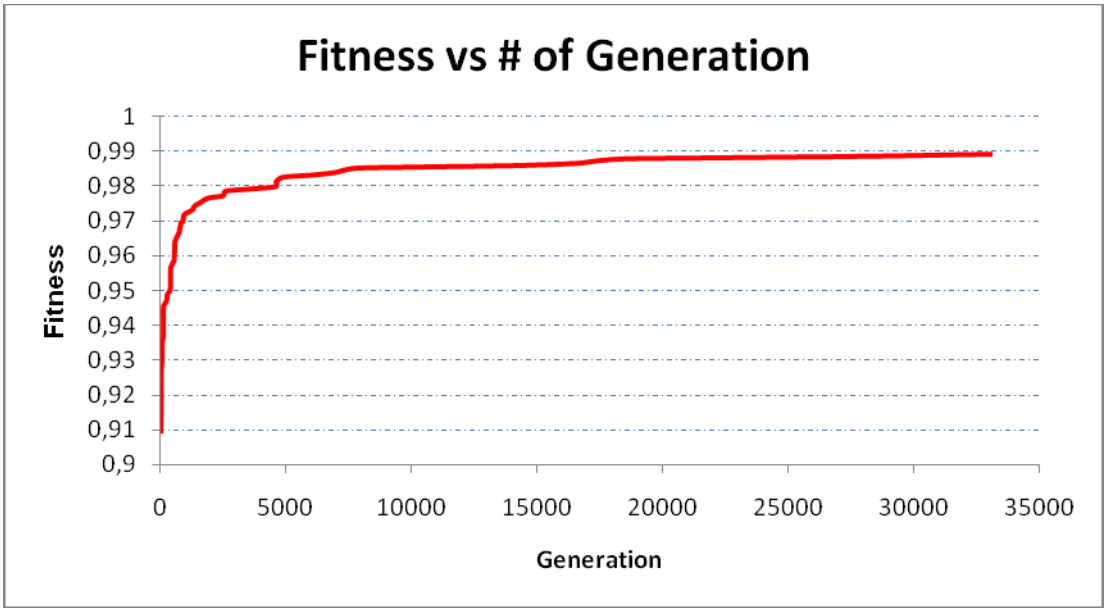

Figure 6.1: Fitness vs. Number of Generation

In Figure 6.2, we present the evolution of proposed genetic algorithm with various probability values of genetic operators. Crossover and mutation probabilities, p_c and p_m respectively, used in experiments are given below:

Experiment#	p_c	p_m
1	0,8	0
2	0	0,8
3	0,8	0,8
4	0,8	0,5

Table 6.1 Crossover and mutation probabilities used in experiments.

The following results can be concluded from these experiments: From experiments #1 and #2, we can conclude that we get better solutions with applying only mutation. However, if we take into consideration all experiments, convergence is higher in the experiments where both genetic operators are used, namely #3 and #4. Even though, we cannot conclude that the higher the probability of the genetic operator, the higher the convergence since experiment #4 has better fitness than experiment #3.

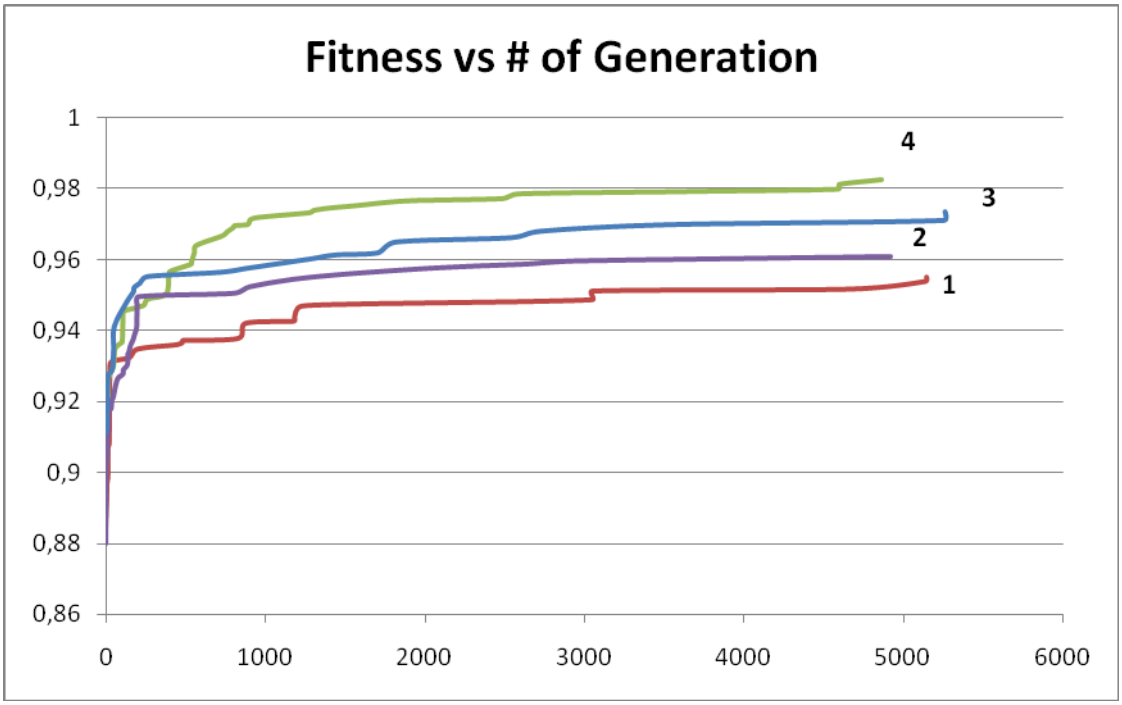

Figure 6.2: Fitness Changing by Probability of Genetic Operators

And finally, we want to analyze the population size effect on the convergence. We tried generations of size 100, 200, 400 and 800. The results can be seen in Figure 6.3. There

is not a uniform convergence related to the population size. Since population size does not affect the genetic algorithm, it is suggested to work with small population sizes in case of long running time.

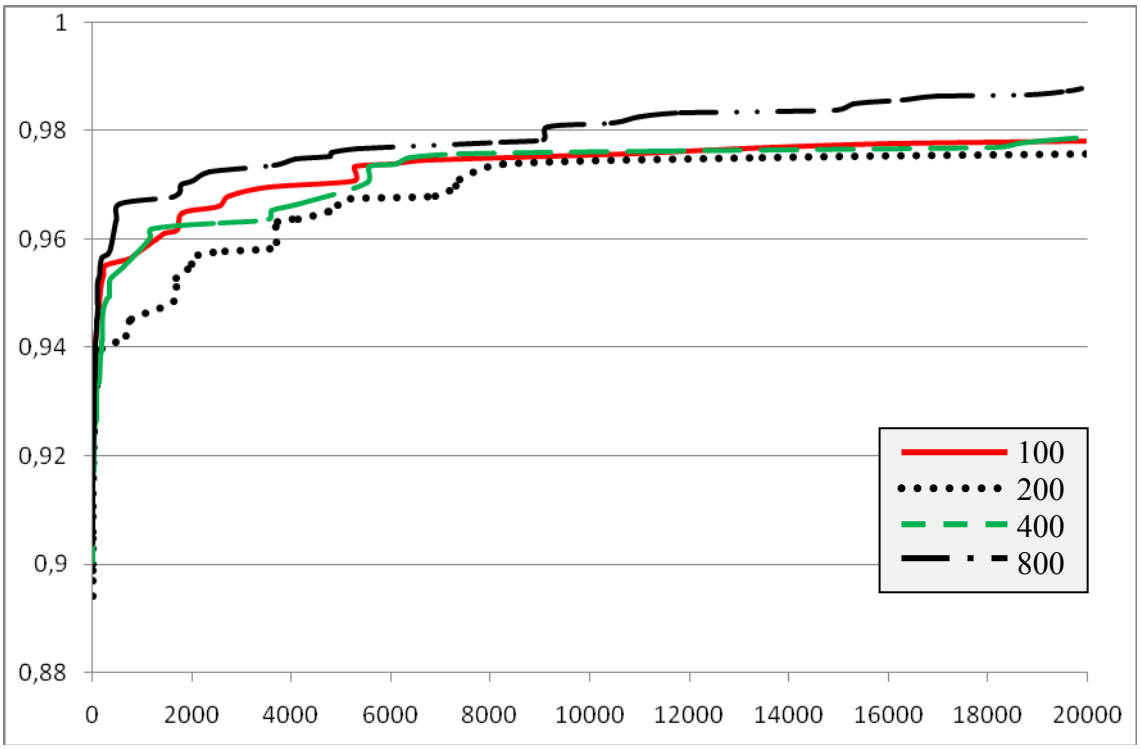

Figure 6.3: Fitness Changing by Population Size

7. CONCLUSION AND DISCUSSION

In general, all metaheuristic search methods are guided with a random decision method. The genetic algorithm is an example for these methods. The genetic algorithm is a probabilistic guided random search since it guides its search based on probability. The major characteristics of the genetic algorithm are randomized search for the solution, crossover and mutation operations and also evolution of the solutions based on the fitness values. Although the method selects the search directions randomly, convergence of the method is satisfied by the genetic operators used in evolution steps.

In this study, we focus on the university timetabling problem whose solution can be obtained with a genetic algorithm. The problem is regarded as scheduling a set of courses into a set of timeslots and a set of rooms without violating hard constraints and satisfying soft constraints as possible as much. The hard constraints must be satisfied in order to obtain a feasible solution. However, due to the random nature of the search and blindness of the genetic operators, traditional genetic algorithms severely violate these constraints many times during the evolution process. This aspect of the operators also makes the convergence time of the algorithm too long. We addressed this problem by introducing modified genetic operators which do not allow violations of any hard constraints, which in turn produce only feasible solutions. Since the proposed method deals with only feasible solutions, efficiency of our algorithm is much better than the classical genetic algorithm.

Another problem with the genetic algorithm is the determination of the best parameter values, such as the population size, mutation rate, etc. It is also very hard to theoretically determine how the genetic operations affect the macroscopic behavior of genetic algorithms such as convergence of solutions. Thus, we try to determine sensitivity of the method to parameters experimentally.

Both crossover and mutation operators are needed in order to prevent the search to be trapped in local minima. Higher probability values for these operators increase the efficiency of the algorithm. And also we realize that increasing the size of the population increases the convergence of the algorithm with respect to the generation number with the cost of high computation.

The advantages and disadvantages of genetic algorithms are not similar to local search methods. Some advantages include: self-guidance, flexibility, simple and straightforward computation, and easy implementation of parallelism. Disadvantages include the chance-dependent outcome and lengthy computation time, yet we may or may not obtain satisfactory solutions. One important consequence of this finding is the realization of applicability of hybrid systems, i.e., genetic algorithms embedded in other methods such as hill-climbing, fuzzy systems, and etc. This strategy may produce more acceptable results and generate practical solutions to the problem in a reasonable limit of time.

Efficiency of the method can also be increased by accounting for blindness of the genetic operators. For example, mutation operations can be performed firstly on the genes which cause dissatisfaction of the soft constraints.

The functionality of the method can be increased by providing a choice of several different good schedules from which the user may choose the best, or by allowing user to make manual adjustment on the timetable.

REFERENCES

Abramson, D., 1991. *Constructing schools timetables using simulated annealing: sequential and parallel algorithms,* Management Science, vol.1, No: 37,pp. 98-113.

Alvarez-Valdes, R., Crespo, E., Tamarit, J.M., 2002. *Design and implementation of a course scheduling system using tabu search.* European Journal of Operational Research 137 (3), 512–523.

Asratian, A.S., de Werra, D., 2002. *A generalised class–teacher model for some timetabling problems.* European Journal of Operational Research 143 (3), 531–542.

Aytug, H., Khouja, M., Vergara, F.E., 2003. *Use of genetic algorithms to solve production and operations management problems: A review*. International Journal of Production Research 41 (17), 3955–4009.

Azimi, Z.N., 2005. *Hybrid heuristics for examination timetabling problem*. Applied Mathematics and Computation 163 (2), 705–733.

Bardadym, V.A., 1996. *Computer-aided school and university timetabling: The new wave.* In: Burke, E., Ross, P. (Eds.),Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153. Springer, Berlin, pp.22–45.

Belligiannis, G., Moschopoulos, C.N., Kaperonis, G.P., Likothanassis, S.D., 2006. *Applying evolutionary computation to the school timetabling problem: The Greek Case*. Computers and Operations Research 35, 1265-1280.

Bhatt, V., Sahajpal, R., 2004. *Lecture Timetabling Using hybrid Genetic Algorithms.* ICISIP 2004. 29-34.

Broder, S., *Final Examination Scheduling*, Comm. of the ACM 7, 494-498.

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

Burke, E., Bykov,Y., Newall,J. P. and Petrovic, S. , 2003, *A Time-Defined Approach to Course Timetabling,* Yugoslav Journal of Operations Research (YUJOR), vol. 13, pp. 139-151.

Carrasco, M.P., Pato, M.V., 2001. *A potts neural network heuristic for the class/teacher timetabling problem,* in: Proceedings of the 4th Metaheuristics International Conference, pp. 139–142.

Carrasco M, Pato M., 2004, *Metaheuristics: computer decision-making, chapter A Potts neural network heuristic for the class/teacher timetabling problem.* Kluwer, Norwell, pp 173–186

Carter, M. W., 1986, *A Survey of Practical Applications of Examination Timetabling Algorithms.* Operations Research 34 193-202.

Carter, M.W., Laporte, G., 1996. *Recent developments in practical examination timetabling.* In: Burke, E., Ross, P.(Eds.), Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol.1153. Springer, Berlin, pp. 3-21

Chaudhry, S.S., Luo,W., 2005. *Application of genetic algorithms in production and operation management: A review.* International Journal of Production Research 43 (19), 4083–4101.

Colorni, A., Dorigo, M., Maniezzo, V. 1990, *Genetic Algorithms and highly constrained problems: The timetable case*. In G.Goos and J. Hartmanis editors.Parallel problem solving from nature. Pp 55-59.Springer Verlag

Davis, L. (Ed) 1991, *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold.

Deris, S., Omatu,S. and Ohta, H., 2000. *Timetable planning using the constraint-based reasoning.* Computers and Operations Research, 27:819-840.

Dorigo, M., Maniezzo,V., Colorni, A., 1991, *Positive feedback as a search strategy,*Technical Report 91-016,Dipartimen to di Elettronica,Politecn ico di Milano, IT.

Ergul, A., 1996. *GA based examination scheduling experience at Middle East Technical University.* In: Burke, E., Ross, P. (Eds.), Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153. Springer, Berlin, pp. 212–226.

Gen, M., Cheng, R., 1997. *Genetic Algorithms and Engineering Design*. Wiley, New York.

Glibovets, N.N., Medvid, S.A., 2003, *Genetic Algorithms Used to Solve Scheduling Problems*, Cybernetics and Systems Analysis,Vol. 39, No.1, 81-90.

Glover, F. , Laguna,M., 1997, *Tabu Search*, Kluwer Academic Publishers.

Gunsdhi, H., Anand,V.J. and Yong ,Y.W. , 1996 *Automated timetabling using an object-oriented scheduler.* Expert System with Applications, 10(2):243-256.

Head, C., Shaban, S., 2005. *A Heuristic approach to simultaneous course/student timetabling.* Computers and Operations Research 34, 919-933.

Hertz, A., 1991, *Tabu search for large scale timetabling problems*, European Journal of Operational Research, vol. 54, pp. 39-47.

Holland, J.H., 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Anna Arbor.

Irene, S., Safaai, D., Hashim, M., Zaiton, S., 2007. *Approaches used in Solving Timetabling Problem: A Review.* Postgraduate Annual Research Seminar.

Isaai, M.T. and Cassaigne, N.P., 2001. *Predictive and reactive approaches to the train-scheduling problem: A knowledge management perspective.* IEEE Transactions on Systems, Man, and Cybernetics -Part C: Applications and Reviews, 31(4):476—484.

Johnson, D., 1993. *A database approach to course timetabling*. Journal of the Operational Research Society, 44(5):425-433.

Karova,M. , 2004. *Solving Timetabling Problems Using Genetic Algorithms*, 27th International Spring Seminer on Electronics Technology, 96-98.

Kanoh, H. , Sakamoto, Y. , 2004. *Interactıve Timetabling System Using Knowledge-Based Genetic Algorithms.* IEEE International Conference on Systems, Man and Cybernetics, 5852-5857

Kirkpatrick, S. Gelatt, C.D. and Vecchi, M.P. (1983). *Optimization by Simulated Annealing,* Science, 220, 671-380.

Kostuch,P., 2005, *The University Course Timetabling Problem with a 3-Phase Approach*, in Practice and Theory of Automated Timetabling (PATAT) V, vol. 3616, Lecture Notes in Computer Science, E. Burke and M. Trick, Eds. Berlin: Springer-Verlag, pp. 109-125.

Krarup, J., and de Werra, D., 1982, *Chromatic Optimisation: Limitations, Objectives, Uses, References*, Euro. J. Oper. Res. 11, 1-19.

Lai, L., Hsueh, N., Huang, L., Chen, T., 2006, *An artificial Intelligence Approach to Course Timetabling,* Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence.

Lewis, R., 2007. *A survey of metaheuristic-based techniques for university timetabling problems,* Springer –Verlag, 167-190.

Merlot,L., Boland ,N., Hughes , B., and Stuckey ,P., 2003, *A Hybrid Algorithm for the Examination Timetabling Problem,* in The Practice and Theory of Automated

Timetabling (PATAT) IV vol. 2740, Lecture Notes in Computer Science, E. Burke and P. D. Causmaeker, Eds. Berlin: Springer-Verlag, pp. 207-231.

Mulvey, J.M., 1982, *A classroom/time assignment model*.European Journal of Operational Research, 9:64-70.

Pongcharoen, P., Promtet W., Yendaree, P., Hicks,C., 2007, *Stochastic Optimisation Timetabling Tool for University course scheduling*, International Journal of Production Economics.

Schmidt, G., and Strohlein, T., 1980, *Timetable Construction--an Annotated Bibliography*, The Computer Journal 23, 307-316.

Smith, K.A. , Abramson, D. and Duke, D. , , 1995. *Hopfield neural networks for timetabling: formulations, methods, and comparative results.* Computer and Industrial Engineering, 44:283--305.

Socha, K. and Samples, M., 2003. *Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art*, in Evolutionary Computation in Combinatorial Optimization (EvoCOP 2003), vol. 2611, Lecture Notes in Computer Science. Berlin: Springer-Verlag, pp. 334-345.

Solotorevsky G., Gudes E., Meisels A., 1994. *RAPS: a rule-based language for specifying resource allocation and timetabling problems.* IEEE Transactions on Knowledge and Data Engineering.

Syarif, A., Yun, Y., Gen, M., 2002. *Study on multi-stage logistic chain network: A spanning tree-based genetic algorithm approach*. Computers & Industrial Engineering 43 (1–2), 299–314.

Thompson, J.M., Dowsland, K.A., 1996. *General cooling schedules for a simulated annealing based timetabling system*. In: Burke, E., Ross, P. (Eds.), Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153. Springer, Berlin, pp. 345–363.

Tripathy, A., 1984, *School Timetabling, A case in large binary integer linear programming,* Management Science, vol.30,pp.1473-1489.

Welsh, D. J. A., and Powell, M. B., 1967, *An Upper Bound for the Chromatic Number of a Graph and its Application to Timetabling Problems*, The Computer Journal 10 , 85-86.

White,G. M., Chan, P.W. , 1979 *Towards the Construction Of Optimal Examination Timetables.* INFOR 17 219-229.

Wren, A., 1996. *Scheduling, timetabling and rostering – a special relationship?* In Lecture Notes in Computer Science: Practice and Theory of Automated Timetabling, E. Burke and P. Ross, editors. Springer, Berlin,Germany, volume 1153, pages 46–75.

Valouxis, C. and Housos, E., 2003. *Constraint programming approach for school timetabling*.Computers and Operations Research, 30:1555-1572.

Voss, S. Martello, S. Osman, I.H. and Roucairol C. (eds.), 1999. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization,* Kluwer Academic Publishers

VITAE

Name and Surname : Mürüvvet Aslı AYDIN

Adress : Bahçeşehir Üniversitesi Fen Edebiyat Fakültesi Çırağan Cd. Osmanpaşa Mektebi Sk. No: 4 – 6 34349 Beşiktaş / İstanbul / Türkiye

Birth Place / Year : Isparta – 1980

Languages : Turkish (native) - English

Elementary School : Isparta Anatolian Trade High School-1995

High School : Isparta Süleyman Demirel Science School-1997

Isparta Ş.A.İ.K. High School-1998

BSc : Boğaziçi University - 2003

MSc : Bahçeşehir University - 2008

Name of Institute : Institute of Sciences

Name of Program : Industrial Engineering

Work Experience : Department of Mathematics and Computer Sciences at Bahcesehir University, Faculty of Art and Sciences ,Teaching Assistant (June 2005 – Recent)

Ugur Education, SAT Team Member (January 2005-April 2005)

Department of Mathematics and Computer Sciences at Kültür University, Faculty of Art and Sciences ,Teaching Assistant (October 2003 – January 2005)