A GENETIC ALGORITHM FOR FINAL EXAM SCHEDULING OF IŞIK UNIVERSITY

Seda YILDIRIM
B.S., Computer Engineering, Girne American University, 2009

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Industrial Engineering

IŞIK UNIVERSITY
2013

IŞIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

A GENETIC ALGORITHM FOR FINAL EXAM SCHEDULING OF IŞIK UNIVERSITY

SEDA YILDIRIM

APPROVED BY:

Assist. Prof. S. Tankut ATAN (Thesis Supervisor)	Işık University	_____
Assoc. Prof. Olcay Taner YILDIZ	Işık University	_____
Assoc. Prof. Çağlar AKSEZER	Işık University	_____

APPROVAL DATE: ..../..../....

A GENETIC ALGORITHM FOR FINAL EXAM SCHEDULING OF IŞIK UNIVERSITY

Abstract

Exam timetabling is a widely encountered scheduling problem at educational institutions. Typically, exam timetabling problems involve some hard constraints and several soft constraints that may vary from one institution to another. One of the soft constraints is that as few students as possible should have more than a predefined number of exams on the same day. At Işık University, if students have more than two exams on the same day they are allowed to ask for makeup exams for the extra exams. While integer programming formulations with other constraints of Işık University could be solved to optimality via commercial solvers, incorporating the daily exam limitation rule proved to be intractable. Hence a genetic algorithm was developed. Using data from several semesters, numerical experiments were conducted to tune the developed genetic algorithm's parameters and test it. The new metaheuristic algorithm was also coded in Java programming language and integrated into finexa, the internally developed exam timetabling software at Işık University.

Keywords: Genetic algorithm, exam timetabling, integer linear model, memetic algorithm

IŞIK ÜNİVERSİTESİ FİNAL SINAV PROGRAMI İÇİN GENETİK ALGORİTMA UYGULAMASI

Özet

Üniversitelerde sıkça karşılaşılan çizelgeleme problemlerden biri sınav haftası programının ayarlanmasıdır. Sınav saatleri ayarlanırken zorunlu ve zorunlu olmayan kısıtlar göz önünde bulundurulur. Zorunlu olmayan kısıtlardan biri, aynı günde istenilenden daha fazla sınava girecek olan öğrenci sayısının olabildiğince az olması kısıtıdır. Işık Üniversitesinde bir günde ikiden fazla finali olan öğrenciler mazeret sınavına girebilme hakkına sahiptir. Işık Üniversitesinde finl programı ayarlanırken dikkat edilen diğer kurallar için tamsayılı programlama ile çözüm üretilebilirken bahsedilen kısıt problemi bu yöntemle çözülemez hale getirdi. Bu nedenle yaklaşık bir yöntem ile çözüm üretme zorunluluğu doğdu. Geliştirilen genetik algoritmayı test ederken dört dönemin verisinden faydalandık. Ayrıca yeni yöntem Java dilinde kodlanarak Işık Üniversitesinde kullanılan sınav çizelgeleme programı finexa arayüzüne entegre edildi.

Anahtar sözcükler: Genetik algoritma, sınav çizelgelemesi, tamsayılı doğrusal programlama, memetik algoritma

Acknowledgements

I would like to thank to my supervisor, Assist. Prof. S. Tankut ATAN for his guidance. I am very grateful to my family and my friends for their encouragements. I also would like to extend my special thanks to Prof. Dr. Mustafa KARAMAN for his precious moral support and help during my graduate studies.

To my family. . .

Table of Contents

List of Tables

List of Figures

List of Abbreviations

GA	Genetic Algorithm
IP	Integer Programming
SAO	Student Affair Office
VBA	Visual Basic for Applications

Chapter 1

Introduction

1.1 Motivation

Exam timetabling is a semesterly activity at universities and other educational institutions. At the end of each semester a final exam for each course of that semester has to be taken by the students during a duration of typically one to two weeks. In Turkey, final exam grades may constitute a significant percentage of students' letter grades. In scheduling final exams, some hard constraints need to be observed. Obviously, each exam needs to be scheduled only once. One is also trying not to schedule exams with common students at the same time. Such assignments result in makeup exams which would mean more work for everybody involved. Thus, it is important to completely avoid such conflicts, if possible. Beyond these common constraints, different institutions may have different limitations often of soft nature, i.e. they should be satisfied as much as possible but some violations can be tolerated.

At Işık University, exam timetabling falls under the responsibility of Student Affairs Office (SAO). Until recently, SAO scheduled exams manually which had many drawbacks. For even modestly sized combinatorial problems, it is impossible for human beings to find good solutions let alone optimal ones. Even finding a feasible solution is very time consuming. Also it is very difficult to go back and find an updated solution should new requests and limitations arise during the

process. SAO spent a couple of weeks for coming up with a basic solution not considering any soft constraints. Given all this, an automated Excel-based system with Visual Basic for Applications (VBA) macros was developed by the industrial engineering department. This system, finexa, considered above-mentioned hard constraints, and some soft constraints. The resulting integer linear model instances could be solved to optimality using a commercial solver, GAMS/CPLEX. Finexa has been successfully used for several semesters since its development.

Işık University regulations allow students to take at most two exams on the same day. Students can ask for makeup exams should their exam schedule require them to take three or more exams on the same day. Incorporating this rule into the mathematical model makes it intractable with commercial solvers and thus it was not considered initially. After a solution that satisfies other constraints was obtained, through facilities provided in finexa, SAO checked for students with three exams on the same day and tried to improve on the schedule as much as possible through trial and error. To overcome this, it was decided to design and implement a heuristic method which will help finexa to directly yield a solution that also takes students with three exams on the same day into consideration. Given its success in implementations for similar problems by other researchers, a genetic algorithm approach was chosen. The newly developed algorithm was coded in Java programming language and also integrated into finexa after numerical experiments and tests.

1.2 Related Work

There is a wide variety of heuristic approaches for solving the examination timetable problem which have been researched. Burke *et al.* [1] prepared a survey for heuristic approaches for the period until 2009. Our literature search covers related articles published after 2009.

Özcan *et al.* [2] performed a comparison between the new candidate solution and the previous solution by applying different heuristic selection methods combined

with the Late Acceptance Strategy. The approach considers that the exams that each student takes must be assigned to different timeslots, the total number of students taking an exam at a timeslot is not allowed to exceed a predetermined capacity, and if a student is scheduled to take two exams in the same day, the exams must not be assigned to successive timeslots.

Sabar *et al.* [3] used a new graph coloring constructive hyper-heuristic. They group the exams based on their enrollment by largest degree, saturation degree, largest colored degree and largest enrollment. Then they build four lists ordered according to exams' difficulty index. The most difficult exam not yet scheduled is scheduled first.

Qu *et al.* [4] hybridized different graph colouring heuristics that construct solutions step by step. Hard constraints avoid students taking two exams at the same time and soft constraints spread the exams taken by students evenly over the timetable.

Abdullah *et al.* [5] hybridized concept of tabu list with memetic algorithm. They utilize different neighbourhood structures and place a structure into a tabu list when it is unable to produce better solutions after crossover and mutation. As hard constraints, they consider that no students should be required to sit two examinations simultaneously.

Malik *et al.*'s [6] work is based on standard tabu search with some modifications. Their constraints are to ensure that no student sits more than one exam at one time (hard constraint), and to spread the exams within the given number of timeslots (soft constraint).

Pillay [7] studied an evolutionary algorithm hyper-heuristic. Two different structures are experimented with where in the first one low-level constructive heuristics are applied sequentially whereas in the second one there is a hierarchical relationship among the heuristics and they are applied simultaneously.

Mansour *et al.* [8] developed an evolutionary heuristic technique based on scatter-search when generating new solutions. They compare their scatter-search technique to several other metaheuristic methods on real data from Lebanese American University and find that it tends to give better results.

Hadjidj and Drias [9] focus on hard constraints where every exam should be scheduled once and no conflicting exams should be scheduled within the same period. They hybridized Greedy Randomized Adaptive Search Procedure and the evolutionary Scatter Search approach. New solutions are improved before replacing others according to their quality and diversity.

Mumford [10] considers two conflicting objectives and provides a multiobjective framework for solving heavily constrained timetabling problems. These objectives are minimizing the length of timetable and spreading exams as much as possible for each student. A greedy algorithm is used for allocating timeslots to examinations. Grouping and reordering techniques are used for improvements. A local search is applied to the best solutions at the end.

Gogos *et al.* [11] investigated a multi-staged approach driven by a Greedy Randomized Adaptive Search Procedure. It involves several optimization algorithms, heuristics and metaheuristics. A construction phase is executed first producing a relatively high quality feasible solution and an improvement phase employs different approaches such as simulated annealing to find better solutions.

Kahar and Kendall [12] compared their constructive heuristic with existing software used in Malaysia Pahang University. Some of their constraints have not been considered before such as the distance of the exam rooms must be close to each other for some exams. These constraints provide additional challenges in defining a suitable model and in developing a constructive heuristic.

Pillay and Banzhaf [13] apply a genetic algorithm with two phases. The first phase focuses on handling the hard constraints and the second phase satisfies soft

constraints as much as possible. The evolutionary process ia guided using domain specific knowledge in the form of heuristics.

El Den and Poli [14] study on a grammer-based genetic programming. The grammar used for producing new generations is based on graph colouring with different slot allocation heuristics.

Kalayci and Güngör [15] also used a genetic algorithm that tries to provide enough time to each of student for studying by maximizing length between difficult exams. Two different genetic algorithm models were developed. First genetic algorithm eliminates infeasible solutions. Second genetic algorithm controls whether the population satisfies the hard constraints or not.

Burke *et al.* [16] investigated Variable Neighbourhood Search approaches. They hybridized this method with a Genetic Algorithm. They also choose appropriate neighbors intelligently.

Ayob *et al.* [17] developed a new software to solve Universiti Kebangsaan Malaysia's exam timetabling problem to substitute the manual process by human schedulers. They proposed a new extended graph colouring heuristic method which produces an intelligent high quality exam timetable for the university. In the intelligent high quality exam schedule, minimizing students with consecutive exams on the same day is considered.

Demeester *et al.* [18] improves the previous solution which is created manually for KAHO Sint-Lieven (Ghent, Belgium) and satisfies all hard and soft constraints. They investigate tournament-based hyper-heuristics and they claim the hyper-heuristic approach produces solutions successfully for complex problems.

Burke *et al.* [19]'s approach is based on hybridizing and deciding which combination of heuristics gives the best result. At the end of their experiments, the Kempe chain move and timeslot swapping heuristic proved to be the best heuristic moves to use in hybridisation. This paper presents a random iterative hyper-heuristic approach which uses improvements by low level heuristics.

Sabar *et al.* [20] utilize a honey bee mating optimization algorithm and report comparable results to other approaches on Carter's problem instances. Alzaqebah and Abdullah [21] use an artificial bee colony search algorithm on the exam timetabling problem with some promising results on some instances used in the literature. There are three categories of bees that is, employed, onlooker and scout bees that communicate with each other in sharing the information about the solutions.

1.3 Outline of Thesis

The rest of the thesis report is organized as follows. Chapter 2 gives information about the exam scheduling processes of Işık University and provides integer programming formulations for the problem. Chapter 3 provides some basic knowledge on genetic algorithms and describes the steps of the developed algorithm. In Chapter 4, numerical experiment results are discussed.

Chapter 2

Integer Programming Formulations

2.1 Final Exam Scheduling at Işık University

All final exams in Işık University are scheduled over a two-week period at the end of the semester, sometimes Saturdays are also included. Each final exam day is divided into three 3-hour slots from 9 am to 6 pm.

Exam scheduling constraints can be divided into hard constraints and soft constraints. Hard constraints cannot be violated. For example, each exam should only be scheduled in one slot and the total enrollment of scheduled exams must not exceed the daily capacity. Soft constraints are those which are not essential but desirable to be satisfied as much as possible. For instance, no student must be scheduled three consecutive exams in a day. Of course specific constraints of exam timetables depends on a university's conditions and capabilities.

When solving an exam timetabling problem we must ensure that hard constraints are not violated and soft constraint costs are minimised. The quality of an exam timetable is measured by the degree it violates the soft constraints and the number of exam conflicts for students.

There are certain rules for scheduling Işık University's final exams. Obviously, there should be no conflicts in the exam schedule of any student. The third slot of each exam day needs to have fewer students because at 6 pm there is limited

shuttle service which is going back to the city. All letter grades have to be in the university's grading system a few days after the exam period finishes. Therefore, the exams of crowded courses are generally scheduled earlier during the exam period so that their instructors have more time to correct the papers. As the university has no physical and proctor capacity constraints, the classroom and proctor assignments are not part of the scheduling process. All classroom and proctor assignments are made after the exam schedule is obtained.

2.1.1 Core Model

Işık University already uses integer programming to deal with certain aspects of its exam timetabling problem which we will call the core model. However, the core model does not impose a limit on the number of exams to be taken by a student on any given day. The core model has the following constraints.

i. Each exam has to be scheduled to one slot only.

ii. The total enrollment in the exams on any day should not exceed the daily student capacity.

iii. Exams should be scheduled to requested slots usually specified as preferred days. For example, it is desired to schedule exams with many students during the first week.

The mathematical model for the exam timetabling problem uses binary variables that control which slot an exam is scheduled.

2.1.1.1 Index sets

C = index set of courses.

D = index set of days.

S = index set of exam slots.

$UDS(c)$ = index set of courses which have undesired slots.

$S(day)$ = index set of slots in day.

2.1.1.2 Parameters

$n_{c\acute{c}}$ = number of students taking both c and $\acute{c}$.

stu_c = number of students in course c.

Cap_{day} = daily student capacity in day.

2.1.1.3 Decision variables

x_{cs} = 1 if the exam of course c is in slot s; 0 otherwise.

$y_{c\acute{c}s}$ = 1 if the exams of courses c and $\acute{c}$ are both in slot s.

e_c = 1 if the course c scheduled undesired s ; 0 otherwise.

2.1.1.4 Formulation

$$min \quad z = \sum_{c} \sum_{\acute{c}>c} \sum_{s} n_{c\acute{c}} y_{c\acute{c}s} + 0.5 \sum_{c} e_c \tag{2.1}$$

subject to

$$\sum_{s} x_{cs} = 1 \quad \forall c \in C \tag{2.2}$$

$$x_{cs} + x_{\acute{c}s} - y_{c\acute{c}s} \leq 1 \quad \forall c \in C, \forall \acute{c} > c \in C, \forall s \in S \tag{2.3}$$

$$\sum_{c} \sum_{s \in S(day)} stu_c x_{cs} \leq Cap_{day} \quad \forall day \in D \tag{2.4}$$

$$\sum_{s \in UDS(c))} x_{cs} - e_c = 0 \quad \forall c \in C \tag{2.5}$$

$$x_{cs} \in \{0, 1\} \quad \forall c \in C, \forall s \in S \tag{2.6}$$

$$y_{c\acute{c}s} \in \{0, 1\} \quad \forall c \in C, \forall \acute{c} > c \in C, \forall s \in S \tag{2.7}$$

$$e_c \in \{0,1\} \quad \forall c \in C \tag{2.8}$$

The aim of the objective function is minimizing of total number of student conflicts in the exam schedule. Equation (2.2) ensures that each course's exam is assigned to only one slot. Eq. (2.3) triggers relevant y variables to become 1 if two courses conflict. The number of exam proctors and the university's daily transportation capacity are limited. For that reason, Equation (2.4) limits the total number of students having exams on a given day. New limits can be specified if the problem becomes infeasible due to these limits. Equation (2.5) is a soft constraint that prevents the exams from being scheduled to undesired slots the penalty cost is added to objective function.The objective function coefficients for the second term was set to 0.5 after some trial-and-error.

2.1.2 Model With No 3 Exam Rule

To model the soft constraints that no student should take three or more exams in a day, we need some additions to the model. The complete formulation is given below. This is the problem for which we are going to provide a heuristic algorithm as it could not be solved with commercial solvers.

2.1.2.1 Index sets

C = index set of courses.

ST = index set of students.

D = index set of days.

S = index set of exam slots.

$UDS(c)$ = index set of undesired slots for course c.

$C(stu)$ = index set of courses taken by student stu.

$S(day)$ = index set of slots in day.

2.1.2.2 Parameters

$n_{c\acute{c}}$ = number of students taking both c and $\acute{c}$.

stu_c = number of students in course c.

Cap_{day} = daily student capacity in day.

$c_{stu,c}$ = 1 if the student stu is taking course c; 0 otherwise.

2.1.2.3 Decision variables

x_{cs} = 1 if the exam of course c is in slot s; 0 otherwise.

$y_{c\acute{c}s}$ = 1 if the exams of courses c and $\acute{c}$ are both in slot s.

$q_{stu,s}$ = 1 if the student stu has an exam in slot s; 0 otherwise.

$w_{stu,day}$ = 1 if the student stu has more than 2 exams on day day; 0 otherwise.

e_c = 1 if the course c scheduled during an undesired slot s ; 0 otherwise.

2.1.2.4 Formulation

$$min \quad z = \sum_{c} \sum_{\acute{c}>c} \sum_{s} n_{c\acute{c}} y_{c\acute{c}s} + 0.5 \sum_{stu} \sum_{day} w_{stu,day} + 0.5 \sum_{c} e_c \tag{2.9}$$

subject to

$$\sum_{s} x_{cs} = 1 \quad \forall c \in C \tag{2.10}$$

$$x_{cs} + x_{\acute{c}s} - y_{c\acute{c}s} \leq 1 \quad \forall c \in C, \forall \acute{c} > c \in C, \forall s \in S \tag{2.11}$$

$$\sum_{c} \sum_{s \in S(day)} stu_c x_{cs} \leq Cap_{day} \quad \forall day \in D \tag{2.12}$$

$$\sum_{s \in UDS(c))} x_{cs} - e_c = 0 \quad \forall c \in C \tag{2.13}$$

$$\sum_{s \in S(day))} q_{stu,s} - w_{stu,day} \leq 2 \quad \forall day \in D, \forall stu \in ST \tag{2.14}$$

$$c_{stu,c}x_{cs} \leq q_{stu,s} \quad \forall c \in C(stu), \forall s \in S, \forall stu \in ST \tag{2.15}$$

$$x_{cs} \in \{0,1\} \quad \forall c \in C, \forall s \in S \tag{2.16}$$

$$y_{c\acute{c}s} \in \{0,1\} \quad \forall c \in C, \forall \acute{c} > c \in C, \forall s \in S \tag{2.17}$$

$$q_{stu,s} \in \{0,1\} \quad \forall stu \in ST, \forall s \in S \tag{2.18}$$

$$w_{stu,day} \in \{0,1\} \quad \forall stu \in ST, \forall day \in D \tag{2.19}$$

$$e_c \in \{0,1\} \quad \forall c \in C \tag{2.20}$$

In the no 3 exam model's objective function, penalty costs for more than two exams in a day are added to the core model's objective function. Equation (2.14) is a soft constraint. It is used to limit each student to at most 2 exams in a day. Equation (2.15) sets q variables to 1 for slots where a student has an exam. q and w are binary variables.

In order to compare the magnitude of the problems, size information of four semester are shown in Table 2.1 and Table 2.2. As shown in the tables the no 3 exam model problem size is quite bigger than the core model.

Table 2.1: Problem sizes of core model

Semester	Number of Constraints	Number of Variables
Spring 2011	462,538	467,456
Fall 2011	494,686	499,772
Spring 2012	612,014	617,986
Fall 2012	568,045	573,495

Table 2.2: Problem sizes of no 3 exam model

Semester	Number of Constraints	Number of Variables
Spring 2011	10,354,662	542,256
Fall 2011	11,855,694	582,852
Spring 2012	13,250,601	705,150
Fall 2012	13,424,690	661,255

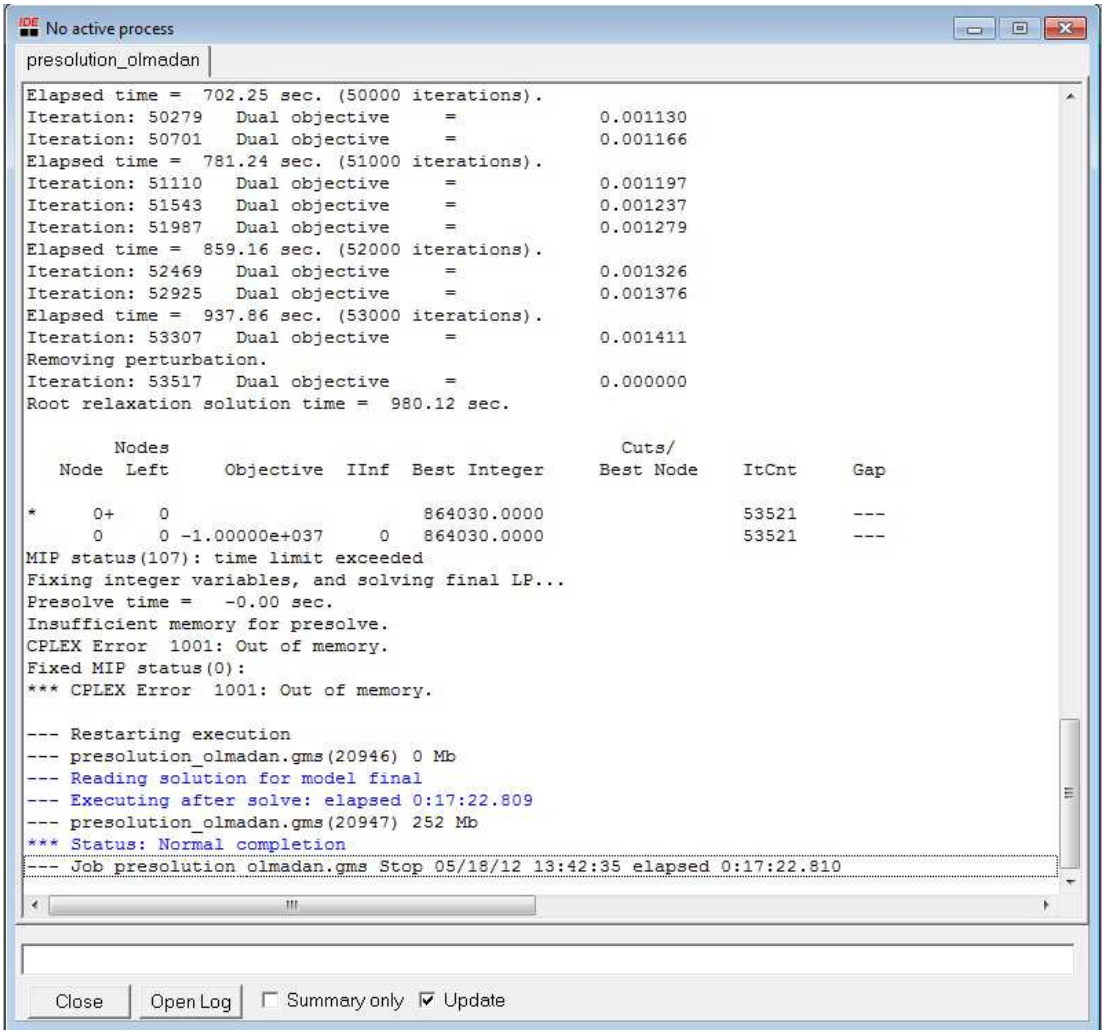

Figure 2.1: GAMS output without pre-solution

The no 3 exam problem instances were also attempted to be solved via GAMS 23.2 using CPLEX as a solver. However, due to memory problems they could not be solved as it can be seen in Figure 2.1. In another experiment, the solution of the core model was given as an initial solution to the no 3 exam model. This approach also did not provide any solution (see Figure 2.2).

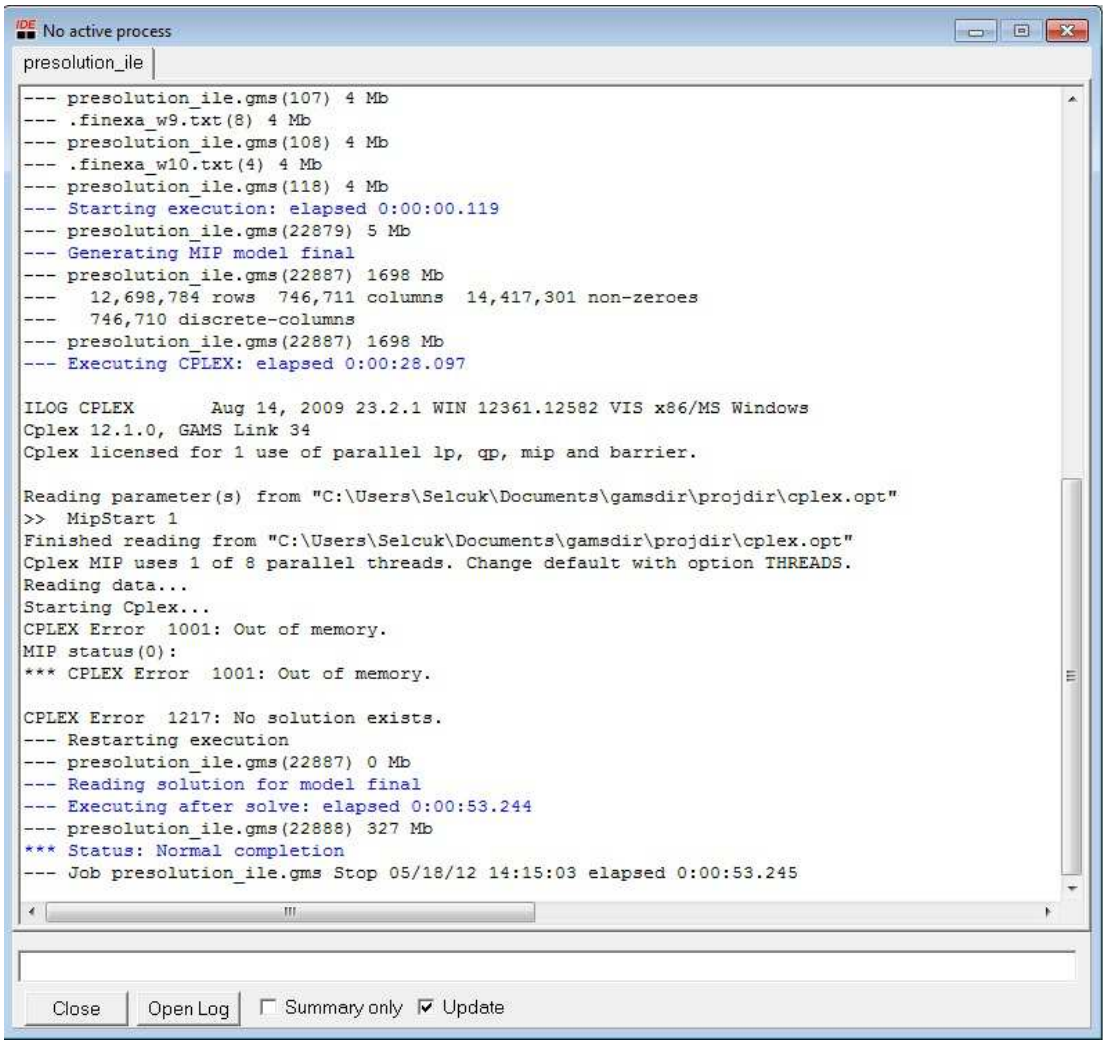

Figure 2.2: GAMS output with pre-solution

Chapter 3

Genetic Algorithm (GA) and Exam Scheduling

3.1 GA

Genetic algorithm (GA) is a heuristic which mimics the national evolution process. What's a heuristic? A heuristic is any technique for finding an approximate solution to a problem when classic methods can not succeed in finding an exact solution.

Genetic algorithms generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, selection, crossover and mutation. Genetic algorithms use the principles of selection and evolution to produce several solutions to a given problem. GA consists of several components. In GA two basic terms which are used frequently are genes and chromosomes. A chromosome refers to each of the individual solutions. A gene is the smallest part of an individual, in other words genes are properties of the solution. Basic steps of a GA are summarized below.

3.1.1 Initialization

The initialization step is used to generate an initial population. This population should be as diverse as possible.

3.1.2 Evaluation

The evaluation function is used to determine the quality of candidate solutions. In GA, the quality of a candidate solution is called fitness. The fitness value is needed for selection and replacement.

3.1.3 Selection

The selection function chooses solutions for recombination. The selection is made by considering solutions' fitness values. There are several commonly used selection methods.

- Roulette-Wheel-Selection: Solutions are selected proportionally to their fitness values. A solution with a higher fitness value has more probability to be selected than a low fitness solution. For example, for a maximization problem consider five solutions with the following fitness values. $f(s_1) = 15, f(s_2) = 20, f(s_3) = 25, f(s_4) = 30, f(s_5) = 35$. The selection probability for each solution can be found using the formula $p(s_i) = \frac{f(s_i)}{\sum_{k=1}^{n} f(s_k)}$. The computed probabilities are as follows. $p(s_1) = \frac{15}{15+20+25+30+35} = \frac{3}{25}, p(s_2) = \frac{4}{25}, p(s_3) = \frac{5}{25}, p(s_4) = \frac{6}{25}, p(s_5) = \frac{7}{25}$.

- Linear-Rank-Selection: This selection method uses the rank of a solution instead of using its fitness value directly. According to their ranks, the selection probabilities are determined. The formula is $s_i = \frac{r(s_i)}{\sum_{k=1}^{n} r(s_k)}$. According to this formula the following selection probabilities are obtain $s_1 = \frac{1}{15}, s_2 = \frac{2}{15}, s_3 = \frac{3}{15}, s_4 = \frac{4}{15}, s_5 = \frac{5}{15}$.

- Tournament Selection: In tournament selection a subset of population is chosen randomly, then the best solution of this subset is selected for recombination. For example, if the individual values which are selected

randomly from population are s_3 and s_4, then s_4 is chosen for recombination because its fitness value is better.

3.1.4 Recombination

In the recombination step two solutions are combined into a new solution. The process combines genes (properties) of selected solutions to construct a new solution.

3.1.5 Mutation

Mutation changes one or more genes (properties) in a chromosome (solution) from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can find a better solution by using mutation.

3.1.6 Replacement

Replacement is the process that combines original population with newly created solutions. There are several replacement schemes, where the most popular are steady state and generational replacement.

- Steady State Replacement: Some of the old solutions become part of the new population. One or more solutions are replaced with better solutions for next generation.
- Generational replacement: In general replacement the new generation replaces the old generation. But in this replacement scheme good solutions can be lost. Elitism is useful for this kind of loss by guaranteing that the best solutions never get eliminated. In elitism, promising solutions are always kept in the population [22].

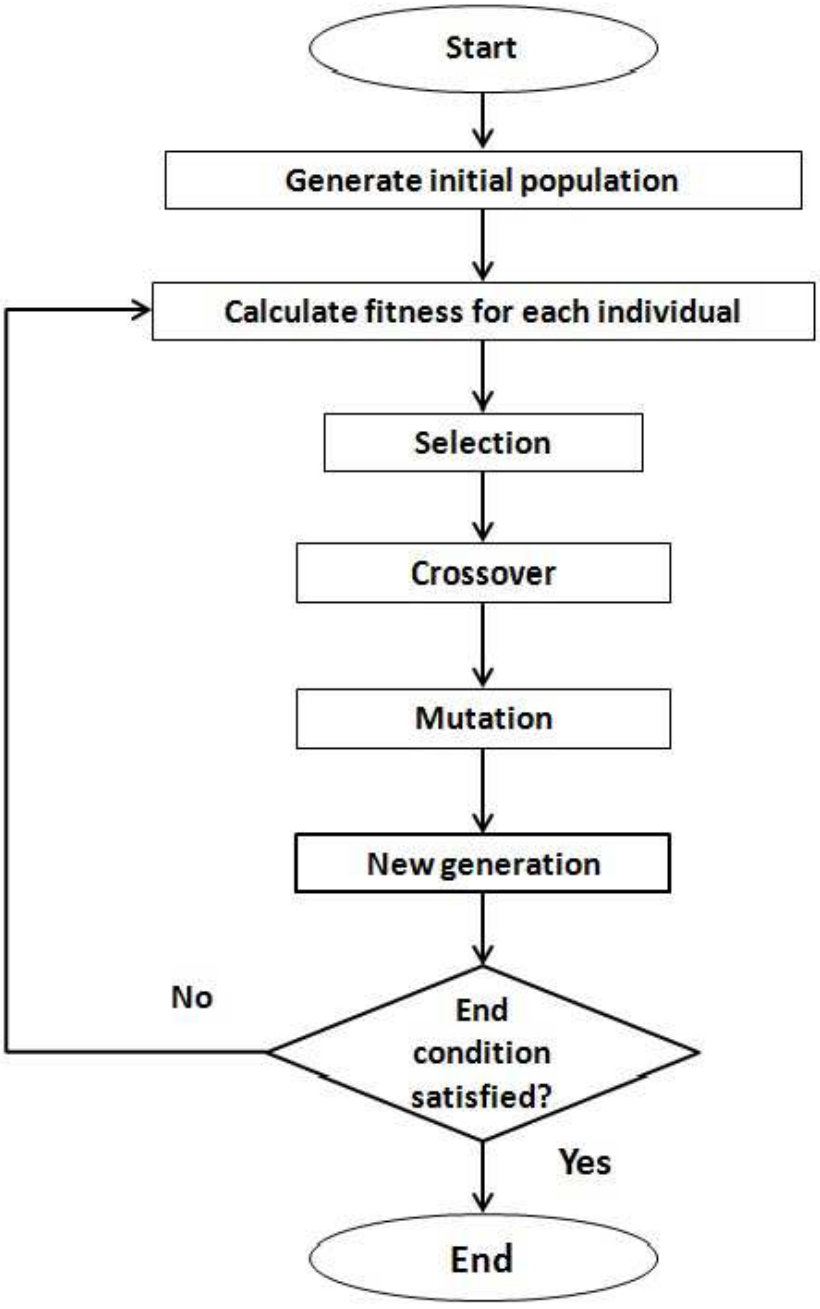

Figure 3.1: GA algorithm

General steps of GA are visually displayed with the flowchart in Fig. 3.1.

3.2 GA for Exam Scheduling

As we mentioned before the commercial solver couldn't find any solution to the problem with no 3 exam rule so we decided to solve it by using GA. GA approach was chosen based on several good results reported in the literature on similar problems. In our solution, in addition to hard and other soft constraints, we specifically focused on decreasing the number of students who have more than two exams in a single day. The applied GA is described below.

3.2.1 Construction Heuristic

In order to generate a feasible initial population, we use the optimal (best) solution of the core model solved via GAMS/CPLEX. The solution coming from the core model ensures that each course is scheduled only once. Hence, GA algorithm does not need to pay attention to this constraint. In order to construct new individuals, random timeslot swaps to this solution are applied without violating any constraints and feasibility. During initial population construction feasibility checks are made so that no student has two exams scheduled at the same time, the daily student capacity does not exceed 1,500, the fixed courses are in their assigned slots, and courses are placed in a desired slot. All these constraints must be considered. Timeslot swaps involve taking complete sets of exams in two randomly chosen timeslots and assigning them to the other timeslot thus changing their timing. Swaps are only made if they are not in violation of the above mentioned constraints. If all constraints are satisfied for a timeslot's courses apart from fix courses then remainining courses can be swapped.

3.2.2 Improvement Algorithm

In order to execute GA, the fitness value of each individual in the initial population is calculated and sorted in increasing order. Fitness values are the total number of students with three exams in a day in each solution. The solution which

has the smallest fitness value gives the best result among the population. After sorting initial population's fitness values, we select some individuals randomly by linear rank selection method. We determined the ratio as 20% for selection and remaining 80% of the next generation comes after applying crossover and mutation operators. Crossover parents are also selected with linear rank selection method again. After crossover, a mutation is applied with 0.20 probability. Initially selected 20% of the old generation and newly generated individuals then constitute the new generation's population. New population is sorted in increasing order according to fitness values and the new solution is obtained while maintaining the size of the population. The process is repeated and stops when the termination criterion is met (in this work the termination criterion is iteration size). We consider following constraints' feasibility when generating initial population, and when applying crossover and mutation.

i. The daily student enrollment cannot exceed 1,500.

ii. The fixed courses cannot be moved from their assigned slots.

iii. Courses should be placed in a desired slot. Preferences are usually expressed as desired days.

iv. No student can take two exams in same slot at the same time.

```
1  Take the optimal CPLEX solution of the core model
2    for 1 to population size
3      Generate other feasible solutions from optimal solution via timeslot swaps
4    end
5    while iteration <= iteration limit
6      Calculate fitness values and sort in increasing order
7      Set aside xx% of the population by using linear ranking
       and place them in selection matrix
8      for 1 to size of crossover population
9        Choose parents for crossover
         Generate new offsprings
10       Apply mutation to new offsprings using mutation rate
11       Store offsprings in crossover matrix
12     end
13     Combine two matrices to build the new generation
14   end while
```

Figure 3.2: GA pseudocode

3.2.3 Solution Representation

Each population member (which represents a feasible solution) is represented as a number of genes that contain information about the timeslot and exams. In Figure 3.3, the numbers are indexes of courses. For example, courses 111, 97, 39, 3 and 2 are scheduled in timeslot S2.

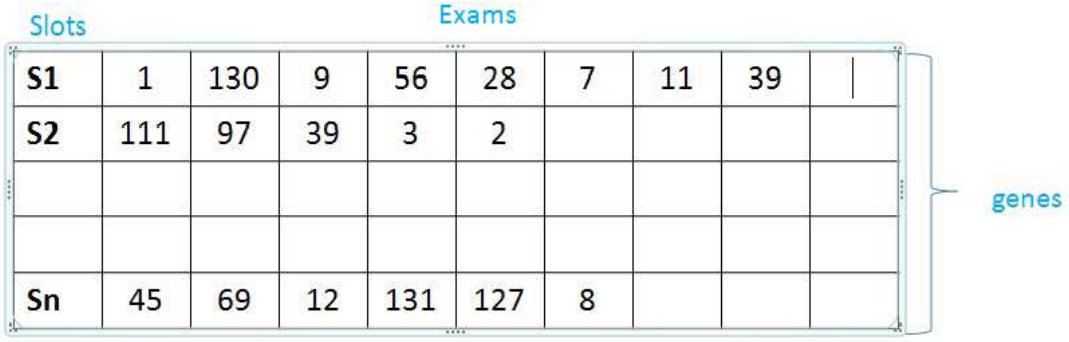

Figure 3.3: Data representation for each chromosome

3.2.4 Crossover and Mutation Operators

In this work, we applied a slot exchange crossover. This crossover operator allows a number of exams (from one timeslot) to be added to another timeslot and vice versa based on a crossover rate (0.8). The crossover parents are selected with linear rank selection method and the exchange slots are decided randomly. We ensure that the feasibility is not violated when we add each course of a timeslot to another timeslot. All hard constraints are checked as in the initial population construction at the beginning. For instance, a course cannot be moved because it is already scheduled in its fixed slot or the move calls for an undesired desired slot for that course. Courses which cannot be moved to their new timeslots are left in their original position. In order to show clearly, how we apply crossover operation, we provide an example where the selected parents and their timeslots to be exchanged are shown in Figure 3.4 and generated new offsprings are shown in Figure 3.5.

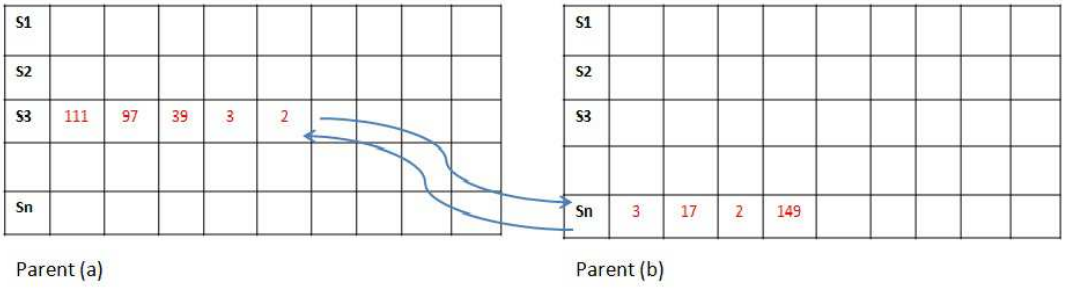

Figure 3.4: Representation of crossover parents

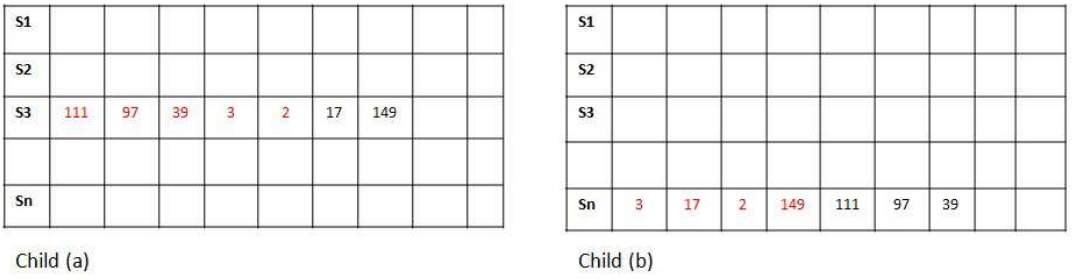

Figure 3.5: Representation of new offsprings

As shown in the Figure 3.4 and 3.5, timeslots S3 and Sn are chosen as parents (a) and (b), all exams from timeslot Sn in parent (b) are added to timeslot S3 in parent (a) to produce child (a). The same process is applied to obtain child (b). The addition process can cause duplication of exams in the same slot. So we check the list of existing exams of the destination slot before inserting the exam. If the exam already exists in the slot, we do not create a copy of it.

The mutation operator selects chromosomes from among the individuals that have gone through crossover with a probability specified by the mutation rate. In mutation, a course (gene) is randomly changed in selected solutions (chromosomes) to a random timeslot.

Chapter 4

Experiments

The optimization model was solved with GAMS 23.2 using CPLEX. GA code was written with JAVA language and the experimental runs were performed on an Intel Pentium 4, 3.20 GHz computer.

4.1 Problem Instances

In our experiments, we used Işık University data from four semesters since Spring 2011. In order to understand how soft constraints effect our solution we also ran our approach on four datasets by relaxing soft constraints. Applying all of the soft constraints increases the number of students with consecutive three exams in a day. But our GA approach still gives better results than SAO's manual process. The parameters used in the algorithm are chosen after preliminary experiments as shown in Table 4.2 (and are comparable with the papers in Abdullah *et al.* [5]). The population size is chosen so that the time to execute the algorithm does stay within an hour for our instances.

Table 4.1: Dataset information

Semester	**# of slots**	**# of courses**	**# of students**	**Density of conflict matrix**
Spring 2011	33	176	1,870	0.08
Fall 2011	30	182	2,077	0.15
Spring 2012	30	193	1,981	0.29
Fall 2012	30	195	2,194	0.14

Table 4.2: Parameter settings

Parameter	**Value**
Population size	100
Crossover rate	0.80
Mutation rate	0.20
Selection mechanism	Linear Rank Selection

4.2 Preliminary Experiments

The optimal solution of the core model which is currently used by SAO forms the basis of the initial population. The GA constructs 99 new members by timeslot swaps. The fitness values of initial population and their sequences are presented in Figure 4.1.

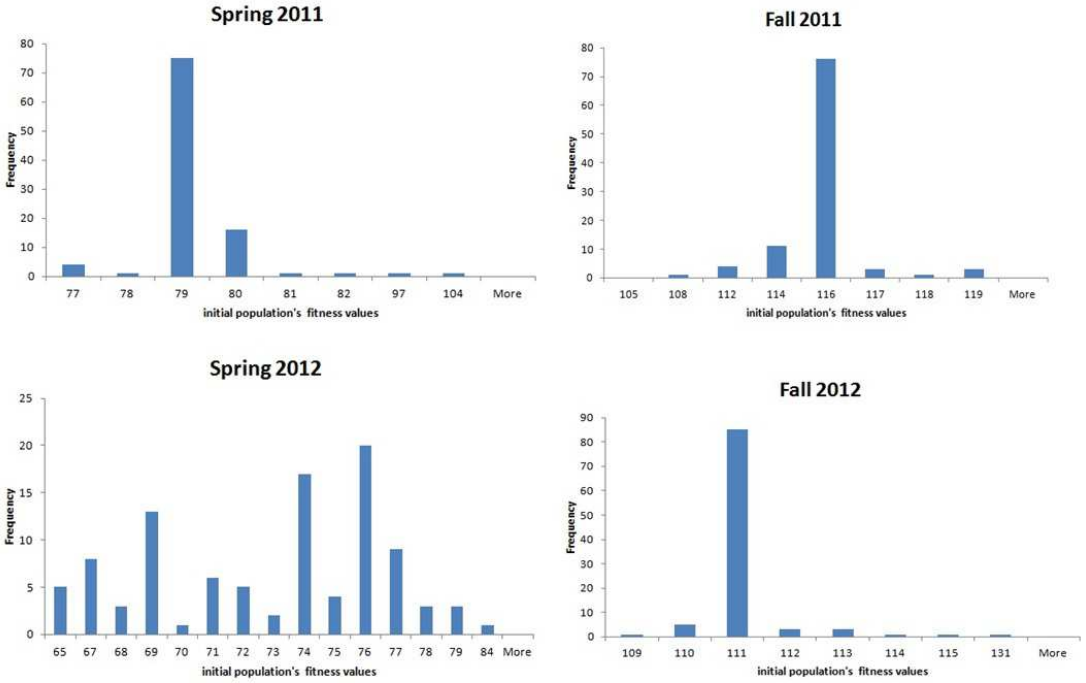

Figure 4.1: Histograms of initial population fitness values

As we look at the experiment results on four datasets, 100 initial solutions and 1,000 iterations generally give better results compared to other trials. The number of students that is scheduled three consecutive exams in a day depends on each semester's specified demands by instructors. The number of students, number of classes but more importantly the desired days and fixed slots requested by lecturers effect and change the solution quality of the problem. The runs show that having more individuals improves the results to some extent. We also see that improvements continue to occur up to 1,000 iterations though they are not very significant after a certain point. Figure 4.2 shows the final results when the GA approach has 100 individuals as initial population with 1,000 iterations.

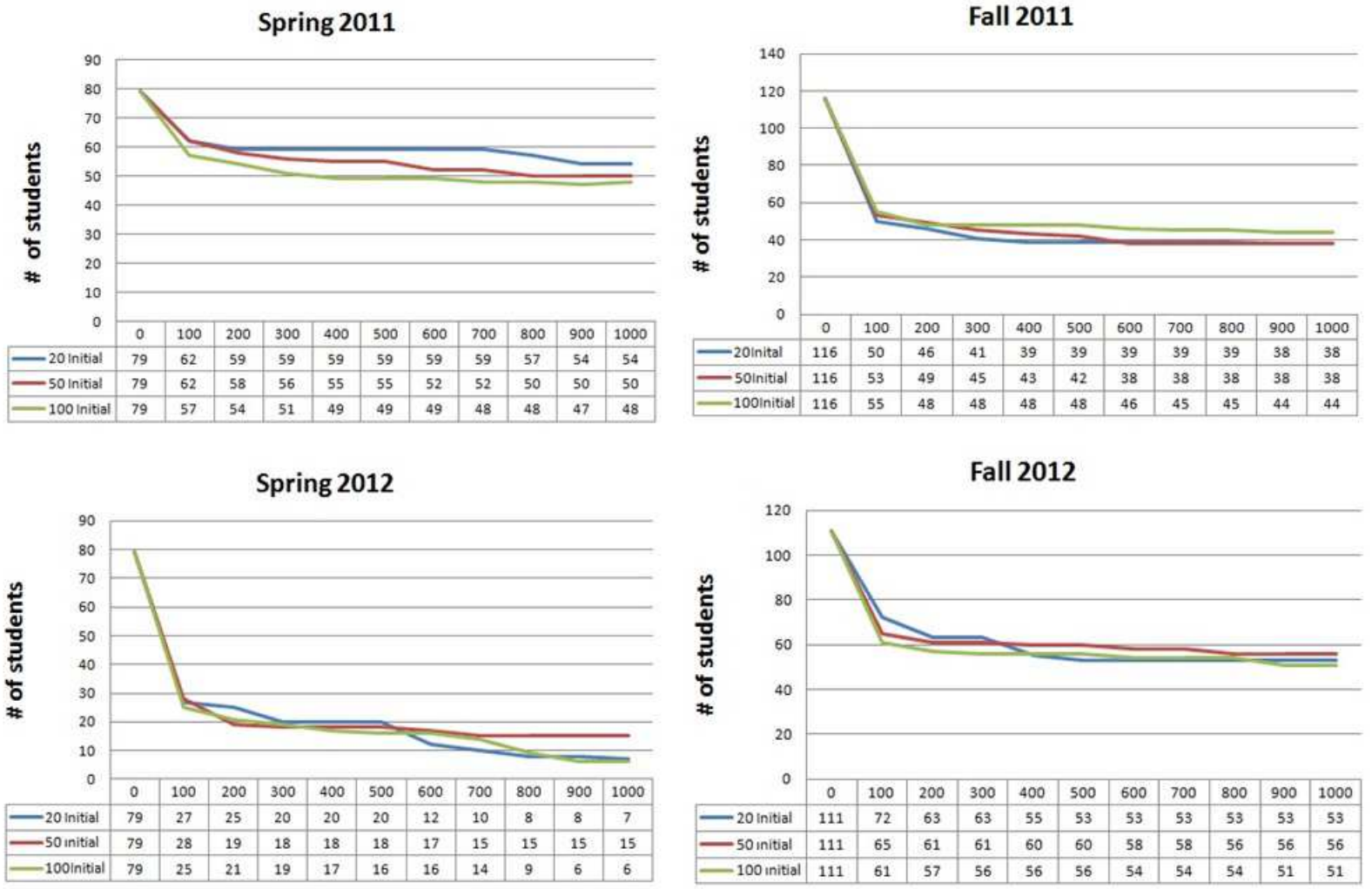

Figure 4.2: Experiments to determine initial population size and iteration limit

Preliminary experiments were conducted to determine crossover and mutation rates to be used in final implementation. Abdullah *et al.* [5] used 0.8 as the crossover rate and 0.04 as the mutation rate. We used those rates for determining initial population size and iteration number. But in order to see how the rates change the final solution, we ran GA with different crossover and mutation rate values separately. The trials and their results are shown in Table 4.3.

Table 4.3: Crossover rate trials and results

	Crossover rates		
Semester	**0.7**	**0.8**	**0.9**
Spring 2011	44	48	40
Fall 2011	43	44	30
Spring 2012	5	6	6
Fall 2012	58	51	56

Table 4.4: Mutation rate trials and results

	Mutation rates		
Data sets	**0.04**	**0.10**	**0.20**
Spring 2011	48	42	39
Fall 2011	44	39	40
Spring 2012	6	8	6
Fall 2012	51	48	49

In Table 4.3 three different crossover rates were tried. The results do not provide a clearcut crossover rate to be set as a bigger rate seems to help in some situations but not in some others. So we chose to use the same rate as in Abdullah *et al.* [5]. For mutation rate, increasing the rate from its initially chosen value of 0.04 and we see that 0.20 improves the results in three out of four cases.

Table 4.5 shows initial fitness values (number of students scheduled 3 exams in a day) and final fitness values with the improvements after applying GA.

Enforcing all constraints increases number of students who have three consecutive exams in a day. These constraints restrict the neighborhood space during initial

Table 4.5: Improvements with GA

Semester	Initial result	Best result	Improvement percentage
Spring 2011	79	39	%49
Fall 2011	116	40	%66
Spring 2012	79	6	%92
Fall 2012	111	49	%55

population construction, crossover and mutation processes, and limit construction of more diverse offsprings. That causes less improvement in the new generation and worse results.

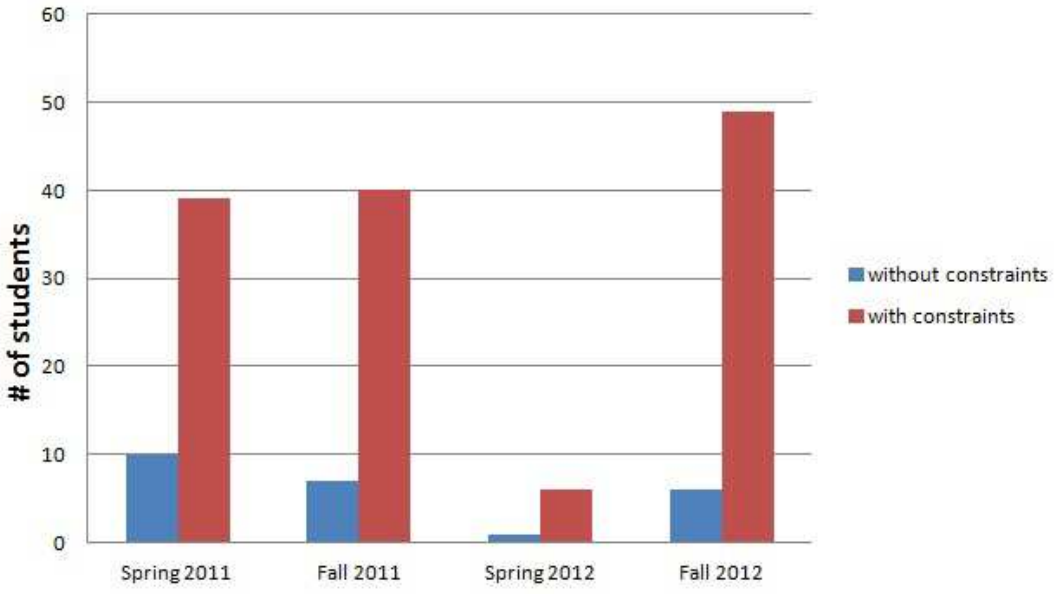

Figure 4.3: Comparison of results with/without day preferences for exams

4.3 Memetic Algorithm

We also investigated how mutation operator can be rendered more intelligent rather than choosing exams randomly and how it would affect the results. In the memetic approach, the exam selection and moving process is done more sensibly. Here, we find more problematic exams taken by students with 3 exams in a day, and then move those in the hope that they will improve the results more. In selecting the exam to be moved, we utilized linear rank selection method again

to give a chance to each of the exams when they are selected rather than always choosing the most problematic exam. Whenever an exam has to be taken as part of a group of three exams in the same day, an exam's score is increased. The selection probability of each exam is based on their rank for calculated points. These points are related to how many students have 3 exams in a day for the current solution. The chosen exam is moved to a randomly determined day with linear rank selection. Destination days' probabilities are based on the total number of students with 3 exams on those days. Days with less students have bigger probability of being chosen. After a day has been chosen, the exam is moved to a random slot. We applied this intelligent mutation process by mixing it with random mutation. Intelligent mutation is applied only during few iterations at the beginning and during remaining iterations random mutation is applied. Results indicate that the memetic algorithm does not provide better solutions. The experiments of memetic approach on four semester is given in Table 4.6.

Table 4.6: Improvements with memetic algorithm

Semester	**Initial result**	**Best result**	**Improvement percentage**
Spring 2011	79	45	%43
Fall 2011	116	44	%62
Spring 2012	79	5	%93
Fall 2012	111	50	%54

Conclusion

The overall goal of this thesis is producing a quality solution for Işık University exam timetabling problem when "no student must have three consecutive exams in a day". Current Excel-based software implemented for SAO does not provide such a solution automatically, and requires manual trial and error. As an exact solution proved to be out of reach for this problem using GAMS/CPLEX, a genetic algorithm was implemented in Java language and integrated to the existing system. Our approach outperforms the manual approach that is used by SAO, and also saves time. One key features of our approach are that the initial population is based on the exact solution of a linear integer model that satisfies other constraints of the university. also there are some differences in applying the GA algorithm.

References

[1] R. Qu, E. K. Burke, B. Mccollum, L. T. G. Merlot, and S. Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, pages 55–89, 2009.

[2] E. Özcan, Y. Bykov, M. Birben, and E. K. Burke. Examination timetabling using late acceptance hyper-heuristics. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 997–1004, Piscataway, NJ, USA, 2009. IEEE Press.

[3] N. R. Sabar, M. Ayob, R. Qu, and G. Kendall. A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence*, 37(1):1–11, 2012.

[4] E. K. Burke R. Qu and B. McCollum. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198(2):392–404, 2009.

[5] S. Abdullah, H. Turabieh, B. McCollum, and P. McMullan. A tabu-based memetic approach for examination timetabling problems. In *Proceedings of the 5th international conference on Rough set and knowledge technology*, RSKT'10, pages 574–581. Springer-Verlag, 2010.

[6] A. Malik, M. Ayob, and A. Hamdan. Iterated two-stage multi-neighbourhood tabu search approach for examination timetabling problem. In *DMO*, pages 141–148. IEEE, 2009.

[7] N. Pillay. An empirical study into the structure of heuristic combinations in an evolutionary algorithm hyper-heuristic for the examination timetabling

problem. In *SAICSIT '10 Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 251–257. ACM New York, 2010.

[8] N. Mansour, V. Isahakian, and I. Ghalayini. Scatter search technique for exam timetabling. *Applied Intelligence*, 34:299–310, 2011.

[9] D. Hadjidj, H. Drias, and M. Bouguerra. A hybrid grasp and scatter search for the exam timetabling problem, 2010.

[10] C. L. Mumford. A multiobjective framework for heavily constrained examination timetabling problems. *Annals OR*, 180(1):3–31, 2010.

[11] C. Gogos, P. Alefragis, and E. Housos. An improved multi-staged algorithmic process for the solution of the examination timetabling problem. *Annals of Operations Research*, 194(1):203–221, 2012.

[12] M.N.M. Kahar and G. Kendall. The examination timetabling problem at universiti malaysia pahang: Comparison of a constructive heuristic with an existing software solution. *Elsevier*, pages 557–565.

[13] N. Pillay and W. Banzhaf. An informed genetic algorithm for the examination timetabling problem. *Applied Soft Computing*, 10:457–467, 2010.

[14] M. B. El Den and R. Poli. Grammar-based genetic programming for timetabling. In *2009 IEEE Congress on Evolutionary Computation, CEC 2009*, pages 2532–2539, 2009.

[15] C. B. Kalayci and A. Güngör. A genetic algorithm based examination timetabling model focusing on student success for the case of the college of engineering at pamukkale university, turkey. *Gazi University Journal of Science*, 25(1):137–153, 2012.

[16] E. K. Burke, A. J. Eckersley, B. Mccollum, S. Petrovic, and R. Qu. Hybrid variable neighbourhood approaches to university exam timetabling. Technical report, 2010.

[17] M. Ayob, A. R. Hamdan, S. Abdullah, Z. Othman, M. Zakree, A. Nazri, K. Abd Razak, R. Tan, N. Baharom, H. Abd Ghafar, R. Md Dali, and N. R. Sabar. Intelligent examination timetabling software. In *Procedia - Social and Behavioral Sciences*, volume 18, pages 600–608, 2011.

[18] P. Demeester, B. Bilgin, P. De Causmaecker, and G. Van Den Berghe. A hyperheuristic approach to examination timetabling problems: Benchmarks and a new problem from practice. *Journal of Scheduling*, 15(1):83–103, 2012.

[19] E. K. Burke, R. Qu, and A. Soghier. Adaptive selection of heuristics for improving exam timetables. *Annals of Operations Research*, pages 1–17, 2012. Article in Press.

[20] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu. A honey-bee mating optimization algorithm for educational timetabling problems. *European Journal of Operational Research*, 216(3):533–543, 2012.

[21] M. Alzaqebah and S. Abdullah. Artificial bee colony search algorithm for examination timetabling problems. *International Journal of Physical Sciences*, 6(17):4264–4272, 2011.

[22] M. Bögl G. Zäpfel, R. Braune. *Metaheuristic Search Concepts*. Springer, Austria, 2010. ISBN 978-3-642-11342-0.

Curriculum Vitae

Seda Yıldırım was born on 17 November 1984, in Bursa. She received his B.S. degree in Computer Engineering in 2009 from Girne American University. She worked as a research assistant at the Department of Computer Engineering of Işık University from 2010 to 2013. The courses which she assisted include Introduction to Programming, Object Oriented Programming, Web Design Programming.

APPENDICES

APPENDIX A

GA Java Code

Data Input and Methods

```
/**
 *
 * @author Seda
 */
import java.io.*;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/*
 *
 * @author Seda
 */
public final class FileReader {

    public Connection conn;
    public Statement stmt;
    private String dataSource;
    public int slot;
    public int iteration;
    public double mutationRate;
    public double crossOverRate;
    public List dersler = new ArrayList<String>();
    public ArrayList<Integer> studentNum = new ArrayList<Integer>();
    public boolean[][] program;
    public boolean[][] kisitli_dersler;
    public boolean[] fixed_dersler;
    public double selection;
    public List ogrenci = new ArrayList<String>();
    public double[] individual;
    public int[] _individual;
    public ArrayList<ArrayList<Double>> finexaSolution = new ArrayList<ArrayList<Double>>();
    List array = new ArrayList<String>();
    ArrayList<String> sorunlu;
```

```
private String file;

public FileReader(String dataSource) throws FileNotFoundException,
 IOException, SQLException {
    this.dataSource = dataSource;
    this.Data1();
    this.readData();
    this.initialTable();
    this.Data();
    this.generateDesire();
    this.fixed();

}

public void Data1() throws FileNotFoundException {
    File dosya = new File("../Heuristic/SorunluDers.txt");

    Scanner sc = new Scanner(dosya);

    sorunlu = new ArrayList<String>();

    while (sc.hasNext()) {

        String sorunluDers = sc.nextLine();


        if (sorunluDers == null) {
            break;
        }
        sorunlu.add(sorunluDers);

    }
    for (int i = 0; i < sorunlu.size(); i++) {
        System.out.println((i + 1) + ".sorunlu ders " + sorunlu.get(i));
    }

}

protected void readData() throws FileNotFoundException, IOException {
    File dosya = new File("../Heuristic/dersler.txt");
    File dosya1 = new File("../Heuristic/ogrencisayi.txt");

    Scanner sc = new Scanner(dosya);
    Scanner sc1 = new Scanner(dosya1);
```

```
        while (sc.hasNext()) {

            String courseCode = sc.nextLine();
            String numberOfStu = sc1.nextLine();
            int numOfStu = Integer.parseInt(numberOfStu);


            if (courseCode == null) {
                break;
            }

            dersler.add(courseCode);
            studentNum.add(numOfStu);

        }


        for (int i = 0; i < dersler.size(); i++) {
        }
        System.out.println("The number of classes " + dersler.size());

        FileInputStream fstream = new FileInputStream("../Heuristic/ogrenci_numaralari.txt");
        FileInputStream fstream1 = new FileInputStream("../Heuristic/alinan_dersler.txt");

        DataInputStream in = new DataInputStream(fstream);
        BufferedReader br = new BufferedReader(new InputStreamReader(in));
        DataInputStream in1 = new DataInputStream(fstream1);
        BufferedReader br1 = new BufferedReader(new InputStreamReader(in1));

        String student;
        String lecture;
        int count;


//Read File Line By Line
        while ((student = br.readLine()) != null) {//stringe ogrencileri aliyor

            // Print the content on the console
            lecture = br1.readLine();//stringe dersleri aliyor

            count = 0;

            for (int i = 0; i < sorunlu.size(); i++) {
                if (lecture.equals(sorunlu.get(i))) {
                    count++;
                }
```

```
            }

            if (ogrenci.contains(student) || count >= 1) {

                continue;
            }

            ogrenci.add(student);

        }

//Close the input stream
        in.close();
        in1.close();

        System.out.println("ogrenci size " + ogrenci.size());


        File dosya2 = new File("../Heuristic/slot.txt");
        Scanner sc2 = new Scanner(dosya2);

        while (sc2.hasNext()) {

            String slotS = sc2.nextLine();

            slot = Integer.parseInt(slotS);
            System.out.println("slot= " + slot);


            if (slotS == null) {
                break;
            }
        }


        File dosya3 = new File("../Heuristic/iteration.txt");
        Scanner sc3 = new Scanner(dosya3);

        while (sc3.hasNext()) {

            String iterationS = sc3.nextLine();

            iteration = Integer.parseInt(iterationS);
            System.out.println("iterasyon  " + iteration);
```

```
            if (iterationS == null) {
                break;
            }
        }


        File dosya4 = new File("../Heuristic/mutation.txt");
        Scanner sc4 = new Scanner(dosya4);

        while (sc4.hasNext()) {

            String mutationR = sc4.nextLine();

            mutationRate = Double.parseDouble(mutationR);
            System.out.println("mutation rate  " + mutationRate);


            if (mutationR == null) {
                break;
            }
        }

        File dosya5 = new File("../Heuristic/crossOver.txt");
        Scanner sc5 = new Scanner(dosya5);
        int b = 0;

        while (sc5.hasNext()) {

            String crossOverR = sc5.nextLine();

            crossOverRate = Double.parseDouble(crossOverR);
            b = (int) (10 * crossOverRate);
            System.out.println("crossover rate  " + crossOverRate);


            if (crossOverR == null) {
                break;
            }
        }
        selection = (double) (10 - b) / 10;

        System.out.println("selection rate = " + selection);
    }

    public void Data() throws FileNotFoundException, IOException
```

```
    {


        FileInputStream fstream = new FileInputStream("../Heuristic/ogrenci_numaralari.txt");
        FileInputStream fstream1 = new FileInputStream("../Heuristic/alinan_dersler.txt");

// Get the object of DataInputStream
        DataInputStream in = new DataInputStream(fstream);
        BufferedReader br = new BufferedReader(new InputStreamReader(in));
        DataInputStream in1 = new DataInputStream(fstream1);
        BufferedReader br1 = new BufferedReader(new InputStreamReader(in1));

        program = new boolean[dersler.size()][ogrenci.size()];
        int row, column, count;

        for (int i = 0; i < program.length; i++) {
            for (int j = 0; j < program[i].length; j++) {
                program[i][j] = false;
            }
        }

        String student;
        String lecture;



//Read File Line By Line
        while ((lecture = br1.readLine()) != null) {
            count = 0;

            for (int i = 0; i < sorunlu.size(); i++) {
                if (lecture.equals(sorunlu.get(i))) {
                    count++;
                }

            }

            if (count >= 1) {
                student = br.readLine();
                continue;

            }

            student = br.readLine();
            row = dersler.indexOf(lecture);
            column = ogrenci.indexOf(student);
```

```
            System.out.println(lecture + " " + row + "and" + column);
            program[row][column] = true;
        }

    }

    public void fixed() throws FileNotFoundException {
        fixed_dersler = new boolean[dersler.size()];

        for (int i = 0; i < fixed_dersler.length; i++) {
            fixed_dersler[i] = true;
        }


        File dosya6 = new File("../Heuristic/fix.txt");
        Scanner sc6 = new Scanner(dosya6);


        while (sc6.hasNext()) {

            String ders = sc6.nextLine();
            if (ders == null) {
                break;
            }
            fixed_dersler[dersler.indexOf(ders)] = false;

            System.out.println("fix ders " + ders + " ");
        }


    }

    public boolean fix(int ders) throws SQLException, FileNotFoundException {

        return fixed_dersler[ders];
    }

    public void generateDesire() throws SQLException, FileNotFoundException, IOException {

        int count;
        kisitli_dersler = new boolean[dersler.size()][slot];

        for (int i = 0; i < kisitli_dersler.length; i++) {
            for (int j = 0; j < kisitli_dersler[i].length; j++) {
                kisitli_dersler[i][j] = true;
            }
```

```
}


File dosya2 = new File("../Heuristic/desired.txt");

Scanner sc1 = new Scanner(dosya2);

while (sc1.hasNext()) {

    count = 0;

    String a = sc1.nextLine();
    String[] b = a.split(",");
    for (int i = 0; i < b.length; i++) {
        b[i].trim();//bosluklar gitti

    }

    for (int i = 0; i < sorunlu.size(); i++) {

        if (b[0].equals(sorunlu.get(i))) {
            count++;
        }
    }
    if (count >= 1) {
        continue;
    }

    int[] s = new int[(b.length) - 1];

    for (int i = 1; i < b.length; i++) {
        Double x = Double.parseDouble(b[i]);
        double y = (double) x;
        s[i - 1] = (int) y;
    }


    int index = dersler.indexOf(b[0]);

    for (int i = 0; i < kisitli_dersler[i].length; i++) {

        kisitli_dersler[index][i] = false;
    }


    for (int i = 0; i < s.length; i++) {
```

```
            int day = (s[i] * 3) - 1;


            for (int j = 0; j < 3; j++) {

                kisitli_dersler[index][(day - j)] = true;

            }

        }


    }


}

public boolean istekler(int data, int slot1) throws SQLException, FileNotFoundException {
    return kisitli_dersler[data][slot1 - 1];
}

public int getLineCount() throws FileNotFoundException {
    String filename = "../GAMS/finexaSolution.txt";
    File file = new File(filename);
    Scanner scanner = new Scanner(file);
    int count = 0;
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        count++;
    }

    return count;
}

public void initialTable() {
    _individual = new int[dersler.size()];

    String filename = "../GAMS/finexaSolution.txt";

    try {

        File file = new File(filename);

        Scanner sc = new Scanner(file);
        String yu;
        System.out.println("finexaSolution yani _individual");
```

```
        for (int i = 0; i < getLineCount(); i++) {



            if (i == 0) {
                yu = sc.nextLine();
                continue;
            }
            String s = sc.nextLine();

            s = s.replaceAll("\"", " ");

            String[] w = s.split(",");

            int orta = Integer.parseInt(w[1].trim());

            _individual[i - 1] = orta;


            System.out.print(_individual[i - 1] + " ");

        }
        System.out.println();

    } catch (Exception ex) {
        System.out.println("File cannot be read!!");
        System.exit(0);
    }

}

public int comparisonTwo(int index1, int index2) {

    int count = 0;
    for (int i = 0; i < program[1].length; i++) {
        if (program[index1][i] && program[index2][i] == true) {

            count++;

        }

    }
    return count;

}
```

```
public int comparisonThree(int index1, int index2, int index3) {
    // int[] conflict = new int[100];
    int count = 0;
    for (int i = 0; i < program[0].length; i++) {
        if (program[index1][i] && program[index2][i] && program[index3][i]) {

            count++;
        }

    }

    return count;
}

public static void BubbleSort(ArrayList<Integer> fitness, ArrayList<Integer> indexes) {
    int temp, temp_1;

    for (int i = 0; i < fitness.size(); i++) {

        for (int j = 1; j < fitness.size() - i; j++) {



            if (fitness.get(j - 1) > fitness.get(j)) {

                temp = (int) fitness.get(j - 1);
                fitness.set(j - 1, fitness.get(j));
                fitness.set(j, temp);
                // System.out.println(j+" ");

                temp_1 = (int) indexes.get(j - 1);
                indexes.set(j - 1, indexes.get(j));
                indexes.set(j, temp_1);
            }

        }
    }

}

public boolean dayCapacity(int slot, int dersIndex, int[] _individual) {


    ArrayList<Integer> depo = new ArrayList<Integer>();
    int sum = 0;
```

```
    if (slot % 3 == 0) {

        for (int i = 0; i < _individual.length; i++) {
            if (_individual[i] == slot || _individual[i] == (slot - 1)
             || _individual[i] == (slot - 2)) {
                depo.add(i);
            }

        }
    } else if (slot % 3 == 2) {

        for (int i = 0; i < _individual.length; i++) {
            if (_individual[i] == (slot - 1) || _individual[i] == slot
             || _individual[i] == (slot + 1)) {
                depo.add(i);
            }

        }
    } else {

        for (int i = 0; i < _individual.length; i++) {
            if (_individual[i] == slot || _individual[i] == (slot + 1)
             || _individual[i] == (slot + 2)) {
                depo.add(i);
            }

        }

    }
    for (int j = 0; j < depo.size(); j++) {

        sum += studentNum.get(depo.get(j));
    }

    if (sum + studentNum.get(dersIndex) <= 1500) {
        return true;
    }


    return false;
}

public boolean hardConst(int[] newGene, int slot, int ders) {
```

```
    ArrayList<Integer> depo = new ArrayList<Integer>();
    int count = 0;

    for (int i = 0; i < newGene.length; i++) {
        if (newGene[i] == slot) {
            depo.add(i);
        }

    }

    for (int i = 0; i < depo.size(); i++) {
        count += comparisonTwo(ders, depo.get(i));
    }
    if (count == 0) {
        return true;
    }




    return false;
}

public int[] crossOver(int randomSlot_1, int randomSlot_2, int[] ch_1, int[] ch_2)
 throws SQLException, FileNotFoundException {
    int count = 0;

    ArrayList<Integer> RandomS1 = new ArrayList<Integer>();
    ArrayList<Integer> RandomS2 = new ArrayList<Integer>();

    for (int i = 0; i < ch_1.length; i++) {
        if (ch_1[i] == randomSlot_1) {
            RandomS1.add(i);
        }
    }
    for (int i = 0; i < ch_2.length; i++) {
        if (ch_2[i] == randomSlot_2) {
            RandomS2.add(i);
        }
    }

    for (int x = 0; x < RandomS1.size(); x++) {
        count = 0;
        for (int y = 0; y < RandomS2.size(); y++) {
            count += comparisonTwo(RandomS1.get(x), RandomS2.get(y));
```

```
            }

            if ((count == 0) && dayCapacity(randomSlot_2, RandomS1.get(x), ch_2)
                    && istekler(RandomS1.get(x), randomSlot_2)
                    && fix(RandomS1.get(x))) {
                ch_2[RandomS1.get(x)] = randomSlot_2;

            }

        }
        return ch_2;
    }

    public void mutation(int[] chromosome) throws SQLException, FileNotFoundException {

        int mutation_count = 0;
        int y = 0;
        while (y != 100) {
            int count = 0;

            ArrayList<Integer> s = new ArrayList<Integer>();
            double a = Math.random() * dersler.size();
            double b = (1 + Math.random() * slot);
            y++;
            int course = (int) a;
            int random_slot = (int) b;


            for (int i = 0; i < chromosome.length; i++) {
                if (chromosome[i] == random_slot) {
                    s.add(i);
                }
            }

            for (int j = 0; j < s.size(); j++) {
                count += comparisonTwo(course, s.get(j));

            }
            if (count == 0 && dayCapacity(random_slot, course, chromosome)
                    && istekler(course, random_slot)
                    && fix(course)) {

                chromosome[course] = random_slot;
                mutation_count++;

                break;
```

```
            }

        }

    }
}
```

APPENDIX B

Genetic Algorithm

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;

/**
 *
 * @author Seda
 */
public class Thesis {

    public static void main(String[] args)
    throws ClassNotFoundException, SQLException,FileNotFoundException, IOException {

        FileReader fr = new FileReader("finexa");

        int[] optimumSol = new int[fr._individual.length];
        int minimumOpt = 1501;

        int[] newGene = new int[fr._individual.length];

        for (int i = 0; i < fr._individual.length; i++) {
            newGene[i] = fr._individual[i];
        }

        int[][] genes = new int[100][newGene.length];

        for (int i = 0; i < newGene.length; i++) {
            genes[0][i] = newGene[i];

        }

        ArrayList<Integer> fitness = new ArrayList<Integer>();
        ArrayList<Integer> indexes = new ArrayList<Integer>();

        int sumOfConflict = 0;//fitness degerleri tutacak
        int daySlot1 = 0, daySlot2 = 0, daySlot3 = 0;
```

```
for (int i = 1; i < genes.length; i++) {

    for (int w = 0; w < 10; w++) {

        double _a = (1 + new Random().nextDouble() * fr.slot);
        double _b = (1 + new Random().nextDouble() * fr.slot);


        int random_slot1 = (int) _a;
        int random_slot2 = (int) _b;

        for (int j = 0; j < newGene.length; j++) {
            if ((newGene[j] == random_slot1) && fr.hardConst(genes[i], random_slot2,j)
                    && (fr.dayCapacity(random_slot2, j, genes[i]))
                    && (fr.istekler(j, random_slot2)) && (fr.fix(j))) {
                genes[i][j] = random_slot2;
        } else if ((newGene[j] == random_slot2) && fr.hardConst(genes[i], random_slot1,
                    && (fr.dayCapacity(random_slot1, j, genes[i]))
                    && (fr.istekler(j, random_slot1)) && (fr.fix(j))) {
                genes[i][j] = random_slot1;
        } else {
                genes[i][j] = newGene[j];
            }
        }

    }
}

File dosya1 = new File("InitialPopulation.txt");
PrintWriter output1 = new PrintWriter(dosya1);
for (int m = 0; m < genes.length; m++) {
    for (int e = 0; e < genes[m].length; e++) {
        output1.print(genes[m][e] + " ");
    }
    output1.println("");
}

output1.close();

File dosya2 = new File("InitialPopulation.txt");
Scanner sc1 = new Scanner(dosya2);


for (int i = 0; i < genes.length; i++) {
    for (int j = 0; j < genes[i].length; j++) {
```

```
            genes[i][j] = (sc1.nextInt());
        }


    }

    File dosya3 = new File("Iterations.txt");
    PrintWriter output3 = new PrintWriter(dosya3);
    File dosya4 = new File("Iterationsnum.txt");
    PrintWriter output4 = new PrintWriter(dosya4);


    /*
     * ana algoritmanin new population ile tekrar donguye girdigi yer
     */
    int heur = 0;
    while (heur < (fr.iteration + 1)) {

        fitness = new ArrayList<Integer>();
        indexes = new ArrayList<Integer>();

        sumOfConflict = 0;//fitness degerleri tutacak
        daySlot1 = 0;
        daySlot2 = 0;
        daySlot3 = 0;

        int[][] slotIndex = new int[fr.slot][];
        int[] slotS;

        for (int i = 0; i < genes.length; i++) {

            slotS = new int[fr.slot];


            for (int z = 0; z < slotS.length; z++) {
                for (int j = 0; j < genes[i].length; j++) {
                    if (genes[i][j] == z + 1) {
                        slotS[z]++;
                    }
                }

            }


            for (int w = 0; w < slotIndex.length; w++) {
                slotIndex[w] = new int[slotS[w]];
```

```
            int y = 0;

            for (int h = 0; h < genes[i].length; h++) {

                if (genes[i][h] == w + 1) {
                    slotIndex[w][y] = h;
                    y++;
                }
            }
        }


        daySlot1 = 0;
        daySlot2 = 1;
        daySlot3 = 2;

        while (daySlot3 <= ((fr.slot) - 1)) {
            for (int x = 0; x < slotIndex[daySlot1].length; x++) {
                for (int y = 0; y < slotIndex[daySlot2].length; y++) {
                    for (int z = 0; z < slotIndex[daySlot3].length; z++) {

                        sumOfConflict += fr.comparisonThree(slotIndex[daySlot1][x],
                         slotIndex[daySlot2][y], slotIndex[daySlot3][z]);

                    }
                }
            }
            daySlot1 += 3;
            daySlot2 += 3;
            daySlot3 += 3;

        }

        fitness.add(sumOfConflict);

        sumOfConflict = 0;

        indexes.add(i);

    }


    if (heur == 0) {
        output3.println("");
        output3.println("The fitness value for " + 0 + "th iteration is "
        + fitness.get(0));
```

```
        output3.println("");

        output4.println("");
        output4.println(+0 + "  " + fitness.get(0));

        output4.println("");

        System.out.println();
        System.out.println("original " + heur + " " + fitness.get(0));
        System.out.println();

    }


    FileReader.BubbleSort(fitness, indexes);

    if (heur > 0) {

        for (int e = 0; e < genes[indexes.get(0)].length; e++) {
            output3.print(genes[indexes.get(0)][e] + " ");

        }

        if (fitness.get(0) < minimumOpt) {
            for (int e = 0; e < genes[indexes.get(0)].length; e++) {
                optimumSol[e] = genes[indexes.get(0)][e];
            }
            minimumOpt = fitness.get(0);
        }
        output3.println("");
        output3.println("The fitness value for " + heur + "th iteration is "
         + fitness.get(0));

        output3.println("");

        output4.println("");
        output4.println(+heur + "  " + fitness.get(0));

        output4.println("");

    }

    if (heur == fr.iteration) {

        File dosyaN = new File("../Heuristic/finexaSolutionHeuristic.txt");
```

```
    PrintWriter outputN = new PrintWriter(dosyaN);
    outputN.println("0.00");
    for (int e = 0; e < optimumSol.length; e++) {
        outputN.println("\"" + (e + 1) + "\"," + "\"" + optimumSol[e] + "\",1.00");
    }

    outputN.close();

}

int[] chromosome_1 = new int[newGene.length];
int[] chromosome_2 = new int[newGene.length];
int[] temp_1 = new int[newGene.length];

double[] LinearRank = new double[genes.length];
double sayi = 0;
double uzunluk_LinearRank = LinearRank.length;
double sum = 0.0;
int count, loop;


for (int i = 0; i < LinearRank.length; i++) {
    sayi += (i + 1);
}


for (int i = 0; i < LinearRank.length; i++) {
    sum += (uzunluk_LinearRank - i) / sayi;
    LinearRank[i] = sum;
}

int uzunluk_selection = (int) (genes.length * fr.selection);

int[][] selection = new int[uzunluk_selection][newGene.length];
loop = 0;

while (loop < selection.length) {

    double select = Math.random();

    int s = 0;

    while (s < LinearRank.length) {
        count = 0;
        if (select < LinearRank[s]) {
```

```
            for (int w = 0; w < selection[loop].length; w++) {

                selection[loop][w] = genes[indexes.get(s)][w];
            }
            count++;

        }

        if (count == 1) {
            break;
        }
        s++;

    }
    loop++;
}


int uzunluk_crossOver = (int) (genes.length * fr.crossOverRate);


if (uzunluk_crossOver % 2 == 1) {
    uzunluk_crossOver += 1;
}


int[][] crossOver = new int[uzunluk_crossOver][newGene.length];
loop = 0;

while (loop < uzunluk_crossOver) {

    double first = Math.random();
    double second = Math.random();

    int i = 0;

    while (i < LinearRank.length) {
        count = 0;
        if (first < LinearRank[i]) {

            for (int w = 0; w < chromosome_1.length; w++) {
                chromosome_1[w] = genes[indexes.get(i)][w];
                temp_1[w] = genes[indexes.get(i)][w];

            }
            count++;
```

```
    }

    if (count == 1) {
        break;
    }
    i++;

}

int j = 0;

while (j < LinearRank.length) {
    count = 0;
    if (second < LinearRank[j]) {

        for (int w = 0; w < chromosome_2.length; w++) {
            chromosome_2[w] = genes[indexes.get(j)][w];
        }
        count++;


    }
    if (count == 1) {
        break;
    }
    j++;

}


double _a = (1 + Math.random() * (fr.slot));
double _b = (1 + Math.random() * (fr.slot));

int random_slot1 = (int) _a;
int random_slot2 = (int) _b;


chromosome_1 = fr.crossOver(random_slot2, random_slot1,
 chromosome_2, chromosome_1);
chromosome_2 = fr.crossOver(random_slot1, random_slot2,
 temp_1, chromosome_2);


//yeni genleri crossOver arrayine ekliyoruz
```

```
                for (int zi = 0; zi < chromosome_1.length; zi++) {
                    crossOver[loop][zi] = chromosome_1[zi];
                    crossOver[loop + 1][zi] = chromosome_2[zi];

                }

                loop += 2;
            }



            for (int i = 0; i < crossOver.length; i++) {

                if (new Random().nextDouble() <= fr.mutationRate) {
                    fr.mutation(crossOver[i]);
                }
            }



            for (int i = 0; i < selection.length; i++) {
                for (int j = 0; j < genes[i].length; j++) {
                    genes[i][j] = selection[i][j];
                }
            }


            for (int i = selection.length; i < genes.length; i++) {

                for (int j = 0; j < genes[i].length; j++) {
                    genes[i][j] = crossOver[i - selection.length][j];
                }
            }
            System.out.println();
            System.out.println("new population " + (heur + 1) + " " + fitness.get(0));
            System.out.println();

            heur++;
        }


        output3.close();
        output4.close();

    }
}
```

APPENDIX C

GAMS Model with No 3 Exam Rule

```
Sets
        c   courses       / 1*183 /
        s   slots         / 1*30 /
        day1(s)           / 1*3 /
        day2(s)           / 4*6 /
        day3(s)           / 7*9 /
        day4(s)           / 10*12 /
        day5(s)           / 13*15 /
        day6(s)           / 16*18 /
        day7(s)           / 19*21 /
        day8(s)           / 22*24 /
        day9(s)           / 25*27 /
        day10(s)          / 28*30 /
        stud   students   /1*1668/;


Alias(c,c1);

Parameters

n(c,c1)  number of students that taking  c and c1 c1>c
/
$include finexa_paramnij.txt
/

stu(c)  number of students in c
/
$include finexa_paramsi.txt
/

cou(stud,c) which student is taking which courses
/
$include finexa_paramtook.txt
/

undsr(c,s) undesired slots for courses
/
$include undesired.txt
/
;

Variables x(c,s), y(c,c1,s),z,q(stud,s),w1(stud),w2(stud),w3(stud),w4(stud),
```

```
w5(stud),w6(stud),w7(stud),w8(stud),w9(stud),w10(stud),e(c);

Binary Variables x, y,q,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,e(c);

Equations obj,assignment(c),conflict(c,c1,s),Capday1,Capday2,Capday3,Capday4,Capday5,Capday6;
Equations Capday7,Capday8,Capday9,Capday10;
Equations soft1(stud),soft2(stud),soft3(stud),soft4(stud),soft5(stud),soft6(stud),soft7(stud);
Equations soft8(stud),soft9(stud),soft10(stud),undesired(c),seda(stud,c,s);

obj ..  z  =e=  sum((c,c1,s), n(c,c1)*y(c,c1,s))+sum((stud),0.5*(w1(stud)+w2(stud)+w3(stud)+
w4(stud)+w5(stud)+w6(stud)+w7(stud)+w8(stud)+w9(stud)+w10(stud)))+0.5*sum((c),e(c));

assignment(c) ..  sum(s, x(c,s))  =e=  1 ;

conflict(c,c1,s)$(ord(c1) gt ord(c)) .. x(c,s) + x(c1,s) - y(c,c1,s) =l=  1 ;

Capday1 .. sum((c,day1), stu(c)*x(c,day1)) =l= 1500;
Capday2 .. sum((c,day2), stu(c)*x(c,day2)) =l= 1500;
Capday3 .. sum((c,day3), stu(c)*x(c,day3)) =l= 1500;
Capday4 .. sum((c,day4), stu(c)*x(c,day4)) =l= 1500;
Capday5 .. sum((c,day5), stu(c)*x(c,day5)) =l= 1500;
Capday6 .. sum((c,day6), stu(c)*x(c,day6)) =l= 1500;
Capday7 .. sum((c,day7), stu(c)*x(c,day7)) =l= 1500;
Capday8 .. sum((c,day8), stu(c)*x(c,day8)) =l= 1500;
Capday9 .. sum((c,day9), stu(c)*x(c,day9)) =l= 1500;
Capday10 .. sum((c,day10), stu(c)*x(c,day10)) =l= 1500;

soft1(stud)..     sum((day1), q(stud,day1))-w1(stud)=l=2;
soft2(stud)..     sum((day2), q(stud,day2))-w2(stud)=l=2;
soft3(stud)..     sum((day3), q(stud,day3))-w3(stud)=l=2;
soft4(stud)..     sum((day4), q(stud,day4))-w4(stud)=l=2;
soft5(stud)..     sum((day5), q(stud,day5))-w5(stud)=l=2;
soft6(stud)..     sum((day6), q(stud,day6))-w6(stud)=l=2;
soft7(stud)..     sum((day7), q(stud,day7))-w7(stud)=l=2;
soft8(stud)..     sum((day8), q(stud,day8))-w8(stud)=l=2;
soft9(stud)..     sum((day9), q(stud,day9))-w9(stud)=l=2;
soft10(stud)..     sum((day10), q(stud,day10))-w10(stud)=l=2;

undesired(c)..       sum(s, undsr(c,s)*x(c,s))-e(c)  =e=  0 ;

seda(stud,c,s)..          cou(stud,c)*x(c,s)=l=q(stud,s);

Model final /all/;
Solve final using mip minimizing z ;
Display z.l,x.l;
```