Ministry of Science and Higher Education of the Republic of Kazakhstan

Suleyman Demirel University

Kuanyshbek Mamay

Solving the exam scheduling problems with Genetic Algorithms

THESIS

Presented in Partial Fulfilment for the

Master of Technical Sciences Degree in Computer Science

(degree code: 7M06102)

Department of Computer Science

Faculty of Engineering and Natural Sciences

Supervisor: **Bek Bazatbek**

Kaskelen 2023

Suleyman Demirel University
Faculty of Engineering and Natural Sciences
Department of Computer Science

Dean of Faculty

Associate Professor

PhD Zhamanov A.

06 2023

Topic of the thesis:

Solving the exam scheduling problems with Genetic Algorithms

Thesis submitted as part of the requirements for the award of the MSc in "7M06102 - Computer Science" SDU, 2021-2023

Head of Department Assistant Professor, PhD Mukash Zh.

Academic Supervisor Bazatbek Bek

Master student Mamay Kuanyshbek

Kaskelen 2023

Ministry of Science and Higher Education of the Republic of Kazakhstan

Suleyman Demirel University

Kuanyshbek Mamay

Solving the exam scheduling problems with Genetic Algorithms

THESIS

Presented in Partial Fulfilment for the

Master of Technical Sciences Degree in Computer Science

(degree code: 7M06102)

Department of Computer Science

Faculty of Engineering and Natural Sciences

Supervisor: **Bek Bazatbek**

Kaskelen, 2023

Suleyman Demirel University
Faculty of Engineering and Natural Sciences
Department of Computer Science

Dean of Faculty

Associate Professor

PhD Zhamanov A.

_____

« _____ »_____
2023

Topic of the thesis:

Thesis submitted as part of the requirements for the award of the MSc in "7M06102 - Computer Science" SDU, 2021-2023

Head of Department _____ Assistant Professor, PhD Mukash Zh.

Academic Supervisor _____ Bazatbek Bek

Master student _____ Mamay Kuanyshbek

Kaskelen, 2023

Declaration

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Kuanyshbek Mamay

2023

Acknowledgements

First of all, I want to thank our creator, the almighty, infinitely good Allah. I want to thank him for giving me the opportunity to write this work. For giving me such intelligence, such knowledge, such health, such wonderful people around me.

I would like to thank my supervisor, Bek Bazatbek, from the bottom of my heart for all the help, support, and guidance he gave me during the whole study process. His knowledge, insightful comments, and commitment to my academic growth have all helped shape this paper.

I want to thank the people who took part in this study from the bottom of my heart for being ready to take part and share their experiences. Their help has been very important in collecting the data needed to answer the study questions and gain useful insights.

I'd like to thank my coworkers and peers for the interesting conversations, helpful feedback, and constant support they have given me. Their efforts and friendship have made the classroom a place where people can work together and where my academic journey has been shaped.

I am very thankful to my family for all the love, support, and confidence they have always shown in me. Their help and understanding have given me the most strength during the highs and lows of this hard road.

Dedication

This dissertation is dedicated to my family, whose unwavering support and encouragement have been the impetus for my pursuit of higher education. I am eternally grateful to my parents, who have sacrificed so much to provide me with every opportunity for success. Your affection, counsel, and confidence in me have shaped the individual I am today.

This work is dedicated to my wife, who has been my rock throughout this voyage. Your unwavering assistance, tolerance, and comprehension have been my constant source of fortitude. This year is not over yet, but it is already very difficult for me and I thought that this year I would not be able to finish my dissertation. But my wife was able to convince me with her faith in me and her energy.Thank you for always having faith in me and being my greatest cheerleader.

This dissertation is also dedicated to my companions, who have supported me throughout the ups and downs of this academic endeavor. Your friendship, laughter, and unwavering support have brought balance and happiness into my life, reminding me of the significance of savoring moments outside of the domain of academia.

Lastly, I dedicate this dissertation to all the instructors and mentors who have inspired and guided me throughout my academic career. Your expertise, enthusiasm, and commitment to education have inspired me to make an impact in my chosen profession.

Abstract

The exam scheduling problem is a complex task faced by educational institutions worldwide. Efficiently allocating exams within limited time slots while considering various constraints, such as student preferences, room capacities, and faculty availability, poses a significant challenge. This dissertation aims to address the exam scheduling problem by leveraging the power of Genetic Algorithms (GAs). Genetic Algorithms are robust search and optimization techniques inspired by the process of natural selection. By employing evolutionary principles, GAs have proven to be effective in finding optimal or near-optimal solutions for a wide range of combinatorial optimization problems. In this study, Ipresent a new application of genetic algorithms to solve the exam scheduling problem, to devise a solution that can be applied to Kazakhstan universities, and to evaluate the performance of GA compared to other existing algorithms commonly used in this field.

Аңдатпа

Емтихандарды жоспарлау мәселесі бүкіл әлем бойынша оқу орындарының алдында тұрған күрделі міндет болып табылады. Студенттердің қалауы, бөлме сыйымдылығы және оқытушылардың қолжетімділігі сияқты әртүрлі шектеулерді ескере отырып, шектеулі уақыт аралығы ішінде емтихандарды тиімді бөлу айтарлықтай қиындық тудырады. Бұл диссертация Генетикалық Алгоритмдердің (ГА) күшін пайдалану арқылы емтиханды жоспарлау мәселесін шешуге бағытталған. Генетикалық алгоритмдер табиғи сұрыптау процесінен шабыттандырылған сенімді іздеу және оңтайландыру әдістері болып табылады. Эволюциялық принциптерді қолдану арқылы ГА комбинаторлық оңтайландыру мәселелерінің кең ауқымы үшін оңтайлы немесе оңтайлы шешімдерді табуда тиімді екенін дәлелдеді. Бұл зерттеуде біз емтиханды жоспарлау мәселесін шешу, Қазақстан ЖОО - на тиімді болатын шешім ойлап табу және осы салада жиі қолданылатын басқа бар алгоритмдермен салыстырғанда ГА - нің өнімділігін бағалау үшін генетикалық алгоритмнің жаңа қолданбасын ұсынамыз.

Аннотация

Проблема планирования экзаменов является сложной задачей, с которой сталкиваются образовательные учреждения во всем мире. Эффективное распределение экзаменов в ограниченные временные интервалы с учетом различных ограничений, таких как предпочтения студентов, вместимость помещений и доступность преподавателей, представляет собой серьезную проблему. Эта диссертация направлена на решение проблемы планирования экзаменов за счет использования возможностей генетических алгоритмов (ГА). Генетические алгоритмы — это надежные методы поиска и оптимизации, вдохновленные процессом естественного отбора. Используя эволюционные принципы, ГА доказали свою эффективность в поиске оптимальных или почти оптимальных решений для широкого круга задач комбинаторной оптимизации. В этом исследовании мы представляем новое применение генетических алгоритмов для решения проблемы планирования экзаменов, разработки решения, которое может быть применено к казахстанским университетам, и для оценки эффективности ГА по сравнению с другими существующими алгоритмами, обычно используемыми в этой области.

Abbreviations

GE - Genetic Algorithm

AI - Artificial Intelligence

RAM - Random Access Memory

IP - Initial Population

PT - Pretty Tables

DNA - Deoxyribonucleic Acid

ML - Machine Learning

Table of Contents

Declaration i

Acknowledgements ii

Dedication iii

Abstract iv

Аңдатпа v

Аннотация vi

List of Abbreviations vii

1 Background and motivations 1

1.1 Introduction 1

1.2 Background 1

1.3 Motivation 2

1.4 Research Objectives 2

1.5 Chapter Summary 3

2 Literature Review 2 4

2.1 Exam Scheduling Problem Overview 4

2.2 Existing Approaches to Exam Scheduling 5

2.2.1 Heuristic-based Approaches 5

2.2.2 Metaheuristic Algorithms 6

2.2.3 Genetic Algorithms 6

2.3 Comparative Analysis of Existing Algorithms 7

3 Genetic Algorithms 8

3.1 Introduction to Genetic Algorithms 8

3.2 Genetic Operators 9

3.2.1 Genetic Operators 9

3.2.2 Selection 9

3.2.3 Corssover 11

3.2.4 Mutation 13

3.3 Genetic Algorithm Parameters 14

3.3.1 Population Size 14

3.3.2 Crossover and Mutation Rates 15

3.3.3 Termination Criteria 16

3.3.4 Physical Fitness Evaluation 17

3.3.5 Elitism 18

4 Problem Formulation 21

4.1 Problem Definition 21

4.2 Exam Scheduling Constraints 22

4.2.1 Hard Constraints 22

4.2.2 Soft Constraints 23

4.3 Encoding Scheme 23

5 Genetic Algorithm Implementation 25

5.1 Design and Implementation Overview 25

5.1.1 Advantages of GA 25

5.1.2 System Architecture 27

5.2 Initialization Stage 27

5.3 Selection Stage: 28

5.4 Crossover Stage 29

5.5 Mutation Stage 30

5.6 Fitness Evaluation 30

5.7 Termination Criteria 31

5.8 Algorithm Flowchart 32

6 Experimental Evaluation **33**
6.1 Experimental Setup . . . 33
6.1.1 Problem Instance . . . 33
6.1.2 Genetic Algorithm Parameters . . . 33
6.1.3 Experimental Data . . . 34
6.1.4 Evaluation Metrics . . . 34
6.1.5 Experimental Procedure . . . 34
6.1.6 Hardware and Software Setup . . . 34
6.2 Dataset Description . . . 35
6.2.1 Data Source . . . 35
6.2.2 Data Preprocessing . . . 35
6.2.3 Dataset Characteristics . . . 35
6.2.4 Data Representation . . . 37
6.3 Experimental Results and Discussion . . . 37
6.3.1 Experimental Results . . . 37
6.3.2 Computational Efficiency . . . 38

7 Conclusion and Future Work **40**
7.1 Future Research Directions . . . 40
7.2 Conclusion . . . 40

Bibliography **42**

Chapter 1

Background and motivations

1.1 Introduction

This chapter provides an overview of the research presented in this dissertation, including its context and motivation. It lays the groundwork for comprehending the significance and relevance of utilizing a Genetic Algorithm to solve the exam scheduling problem. The chapter begins with a review of the exam scheduling problem's background information, followed by an analysis of the motivation behind this research.

1.2 Background

Exam scheduling is a challenging endeavor for educational institutions worldwide. The creation of conflict-free and efficient schedules for a large number of examinations, taking into account various constraints and preferences, represents a formidable obstacle. The exam scheduling problem entails allocating resources such as examination rooms, invigilators, and time periods to various exams, while satisfying constraints such as avoiding conflicts for students taking multiple exams and adhering to academic calendar-imposed time constraints.

Various approaches and algorithms have been proposed in recent years to address the exam scheduling problem. These methods consist of constraint program-

ming, mathematical optimization, and heuristic methods. While these methods have made significant contributions, they frequently suffer from computational complexity, suboptimal solutions, and the inability to accommodate real-world constraints. Consequently, alternative approaches that can surmount these limitations and provide more efficient and effective solutions are required [1].

1.3 Motivation

The motivation for this study is the inadequacy of existing solutions to the problem of exam planning. Due to the computational complexity of the problem, traditional methods can make it difficult to manage large-scale instances[2]. In addition, they may generate sub-optimal schedules that do not maximize the use of available resources or properly address student preferences. Therefore, there is a strong need for an alternative strategy that can address these limitations and improve productivity and efficiency.

Natural selection and genetic processes motivate genetic algorithms (GAs), a subclass of evolutionary algorithms. GAs have shown promise in solving combinatorial optimization issues by developing a population of candidate solutions utilizing the principles of selection, crossover, and mutation. Due to their inherent parallelism and global search capabilities[3], GAs are well suited to complex tasks such as exam planning.

This research aims to develop a genetic algorithm-based exam scheduling solution for universities in Kazakhstan. Using the power of GA, this study aims to overcome the limitations of traditional methods and provide an efficient and high quality exam scheduling. The proposed algorithm will take into account various constraints and preferences, optimize resource usage, and generate conflict-free schedules.

1.4 Research Objectives

The objectives of the dissertation research are the following:

Create a genetic algorithm based solution to the exam scheduling problem. Evaluate the performance and efficiency of the proposed genetic algorithm approach compared to other existing methods. Determine the advantages and disadvantages of the GA-based method in terms of solution quality, computation time, and scalability. Studying the use of genetic algorithms for solving combinatorial optimization problems in the context of exam planning[4].

1.5 Chapter Summary

This chapter provided a brief overview of the thematic research, including its scope and motivation. It highlighted challenges in experimental design and limitations of existing methods. That focusing on the need for alternative approaches that could overcome these limitations informed the rationale for this study. The aims of the research were enumerated, and the tone of the subsequent chapters was set.

Chapter 2

Literature Review 2

2.1 Exam Scheduling Problem Overview

Exam scheduling is a complex optimization problem in which exams are assigned to a predetermined schedule while taking into account a number of constraints and objectives. The objective is to design an effective and equitable exam schedule that minimizes scheduling conflicts and satisfies all requirements.

In this scenario, it is necessary to schedule a series of examinations within a limited amount of time, typically a few days or weeks. Each exam has specific requirements, such as duration, number of students taking the exam, and resources, such as classrooms and proctors. In addition, there are a number of constraints to consider, including:

Student Constraints: Ensuring that no student has exams that intersect or too many exams on a single day.

Room Constraints: Assigning classrooms that can accommodate the number of students taking the exam and any special needs, such as apparatus or accessibility.

Instructor Constraints: Preventing scheduling conflicts for instructors who may teach multiple courses or have other obligations.

Resource Constraints: Managing the availability and allocation of resources, such as security personnel, specialized apparatus, and room characteristics.

Time Constraints: Taking into account any exam scheduling restrictions, such as avoiding certain time slots or daytime pauses.

The exam scheduling problem is known to be a highly constrained and combinatorial optimization problem, which makes it computationally difficult to find the optimal solution among all possible combinations. As a metaheuristic optimization technique, genetic algorithms have been extensively used to address this issue by generating and evolving populations of potential solutions.

Researchers and practitioners utilize genetic algorithms to discover an exam schedule that minimizes conflicts, maximizes resource utilization, and satisfies all constraints. The algorithm improves the quality of solutions by employing selection, crossover, and mutation operators over multiple generations to a population of schedules.

Because it directly affects the equity and effectiveness of the examination process, the exam scheduling issue is of great practical importance in educational institutions. Effective scheduling can reduce student anxiety, avert conflicts, and maximize resource utilization. In order to address the complexities of the exam scheduling problem and enhance the quality of generated schedules, researchers continue to investigate and develop novel approaches. [5].

2.2 Existing Approaches to Exam Scheduling

2.2.1 Heuristic-based Approaches

A heuristic is an approach or method for solving problems that seeks a satisfactory solution without assuring optimality. It is a practical and effective method that employs experience, intuition, and rules of thumb to guide the solution-finding process. Heuristics are frequently employed when the problem is complex or when finding an optimal solution in a reasonable amount of time is computationally impossible. Approaches based on heuristics provide a solution of acceptable quality that may not be optimal but is acceptable for practical purposes[6]. Typically, these methods rely on local search algorithms that modify an initial solution iteratively until a satisfactory solution is found. In the literature review

titled "Optimizing Exam Schedules: A Literature Survey" [7], the Simulated Annealing algorithm, Tabu Search, Ant Colony Optimization, and Particle Swarm Optimization are discussed as heuristic-based solutions for the exam scheduling problem.

2.2.2 Metaheuristic Algorithms

A metaheuristic is a problem-solving strategy at a higher level that guides the search for diverse and adaptable solutions. It is an algorithmic framework that can be applied to a wide range of optimization problems[8]. Metaheuristics are inspired by natural or social phenomena and frequently employ analogies between biological and physical systems. The goal of these algorithms is to locate high-quality solutions by iteratively refining a set of candidate solutions as they traverse the global search space. Metaheuristics are frequently employed to solve difficult or impossible optimization problems involving complex variables.

Metaheuristic algorithms, including Genetic Algorithms (GAs), are methods for optimizing populations based on natural processes. GAs generate new solutions from a population of candidate solutions using selection, crossover, and mutation operators. The survey paper [7] also mentioned genetic algorithms as a metaheuristic exam scheduling solution.

2.2.3 Genetic Algorithms

Genetic Algorithms (GAs) are a subset of evolution programs that are founded on the Darwinian principles of natural selection and random mutation. John Holland formalized them in 1975, and their prevalence has increased since then, particularly for solving problems with a large irregular search space of potential solutions, as described by Colorni et al., Rawat and Rajamani [9, 10]. Maintaining a population of feasible timetables. The most optimal timetables are chosen as the premise for the subsequent iteration or generation. For optimal results, fundamental operators such as selection, mutation, and crossover are utilized. A program written in C was used to implement and control the initialization of a population, the evaluation, and the genetic operators. Each chromosome

would be large and contain an allele for scheduling each instruction. Each class would be assigned a room and time period by the GA, whose suitability would be determined by the number of constraint violations.

2.3 Comparative Analysis of Existing Algorithms

You need to do a comparative analysis to evaluate the performance and effectiveness of various exam planning algorithms. In a review article [7], I compared several algorithms such as simulated annealing, taboo search, ant colony optimization, particle swarm optimization, and exact methods, discussed their advantages, disadvantages, and applications in solving the exam planning problem. This comparative analysis identifies the most promising algorithms and gives an idea of the quality of their solutions, computation time and ease of implementation. See Appendix A.

It presented an overview of the problem and discussed existing solutions such as heuristic-based approaches and metaheuristic algorithms, including genetic algorithms. The review article [7] was a valuable resource as it discussed various algorithms and their applications for solving the exam scheduling problem. And based on this literature review, I decided to solve this problem with a Genetic algorithm. Because of its high efficiency and ease of implementation

Chapter 3

Genetic Algorithms

3.1 Introduction to Genetic Algorithms

A genetic algorithm is a heuristic search algorithm used to solve optimization and modeling problems by random selection, combination and variation of the desired parameters using mechanisms similar to natural selection in nature[11]. It is a type of evolutionary computation that solves optimization problems using natural evolution methods such as inheritance, mutation, selection, and crossing over.

Genetic Algorithms(GA), at their essence, simulate the process of natural selection, in which individuals with favorable characteristics are more likely to survive and reproduce, passing on their genetic information to the next generation[8]. Similarly, in a GA, a population of candidate solutions represents potential solutions to the problem at hand. By iteratively applying genetic operators such as selection, crossover, and mutation, the GA evolves and improves the population over successive generations.

The primary benefit of Genetic Algorithms is their capacity for global search space exploration. They are not limited by local optimums and can identify promising regions of the solution space that other methods might overlook. This global exploration makes GAs ideal for intricate optimization problems where finding the optimal solution is difficult[12]. Metaheuristic algorithms, such as Genetic Algorithms (GAs), are techniques for optimizing populations based on

natural processes. GAs operate on a population of candidate solutions and generate new solutions using selection, crossover, and mutation operators. The survey paper [7] also mentioned the use of genetic algorithms as a metaheuristic solution to the exam scheduling issue. In exam scheduling problems, a genetic algorithm can be used to determine an efficient and conflict-free schedule that satisfies multiple constraints such as room availability, student intersections, and instructors. By encoding the exam scheduling problem into an appropriate representation, such as a chromosome or a sequence of genes, a GA can efficiently manipulate and evolve potential schedules[13].

3.2 Genetic Operators

3.2.1 Genetic Operators

Initialization is a crucial step in the implementation of Genetic Algorithms (GAs). It involves creating an initial population of candidate solutions, also known as individuals or chromosomes, for the problem being solved. The quality and diversity of the initial population can significantly impact the performance and convergence of the GA[14]. See Figure 3.1.

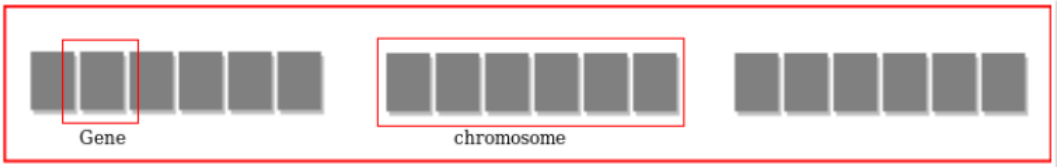

Figure 3.1: Representation of population

3.2.2 Selection

Selection is a key genetic operator in Genetic Algorithms (GAs) that determines which individuals from the population will be chosen for reproduction and contribute their genetic information to the next generation. The selection process is analogous to the natural process of survival of the fittest, where individuals with better fitness values have a higher probability of being selected[15].

There are various selection methods available in GAs, each with its own characteristics and advantages. Some commonly used selection methods include:

1. Roulette Wheel Selection: This technique, also known as fitness proportionate selection, allocates a selection probability to each individual based on their fitness value. Individuals with greater fitness have a greater chance of being selected, while those with lesser fitness still have a chance of being chosen. Similar to a roulette wheel, the selection probability is proportional to the fitness value, hence the name[16]. An example of this is shown in Figure 3.2 and Figure 3.3

Individuals	Fitness Value	Selection Probability	Selection Probability (%)
Individual 1	720	720/3000 = 0.24	0.24
Individual 2	630	630/3000 = 0.21	0.21
Individual 3	390	390/3000 = 0.13	0.13
Individual 4	540	540/3000 = 0.18	0.18
Individual 5	240	240/3000 = 0.08	0.08
Individual 6	480	720/3000 = 0.16	0.16
Total	3000		100

Figure 3.2: Roulette wheel method

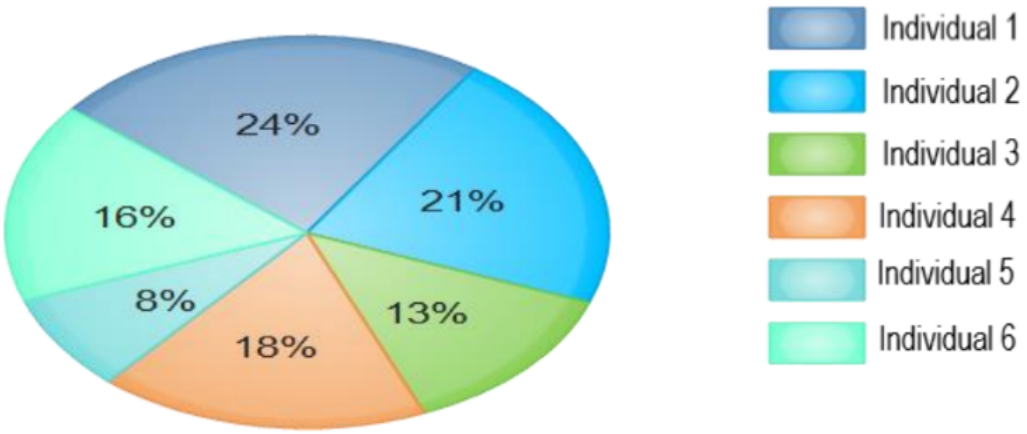

Figure 3.3: Roulette wheel pie chart

2. Tournament Selection: In tournament selection, a random subset of the population, known as a tournament, is selected. Individuals in the tournament compete against one another, and the individual with the highest fitness value is chosen for reproduction. Tournament selection enables a balance

between exploration and exploitation by providing a method for selecting both optimal and potentially suboptimal solutions[17]. An example of this method (the number of individuals entering the tournament is determined as 3) is shown in Figure 3.4.

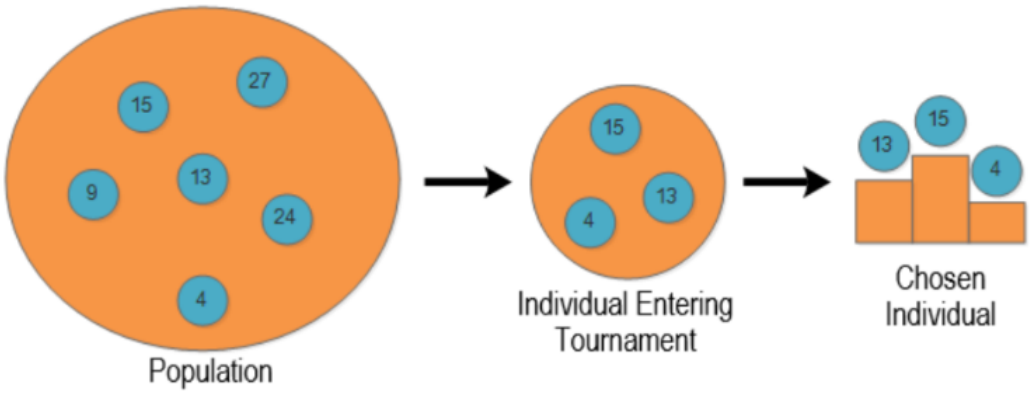

Figure 3.4: Tournament selection method

3. Rank-Based Selection: In rank-based selection, each individual is assigned a rank based on its fitness value relative to the rest of the population. The probability of selection is then determined based on the rank as opposed to the actual fitness value. This strategy reduces the influence of extreme fitness levels and promotes diversity among the selected individuals [18].

The choice of selection method is determined by the nature of the problem, the size of the population, and the intended balance between exploration and exploitation. Different selection methods exert distinct selection pressures, which influence the GA's diversity and convergence properties.

3.2.3 Corssover

In Genetic Algorithms (GAs), crossover is a fundamental genetic operator that replicates the process of reproduction and genetic recombination in biological systems. It is the process of combining the genetic information of two parents to produce progeny with potentially enhanced characteristics. Crossover is essential for investigating the search space and fostering the exchange of genetic material in order to generate diverse and potentially superior solutions[19].

In the context of the exam scheduling dilemma, crossover is the process of combining the exam assignments or time slots from two parent schedules to generate new offspring schedules. The crossover operator specifies the principles governing the exchange of genetic material between parents in order to produce viable offspring.

There are several types of crossover operators commonly used in GAs, including:

1. One-Point Crossover: In this one-point intersection, a random intersection point is chosen and the tails of its two parents are swapped to obtain new original elements. An example of this is shown in Figure 3.5

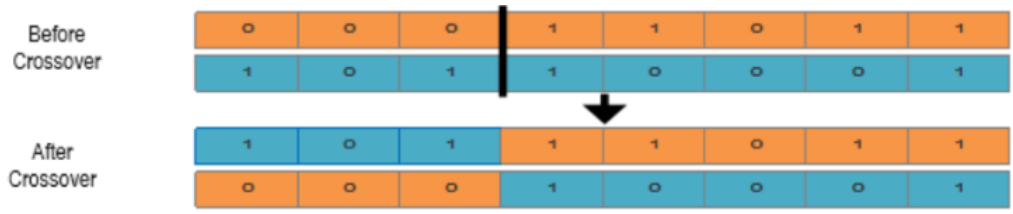

Figure 3.5: One-Point Crossover

2. A multi-point crossover is an extension of a single-point crossover that swaps alternating segments to produce fresh output springs. An example of this is shown in Figure 3.6.

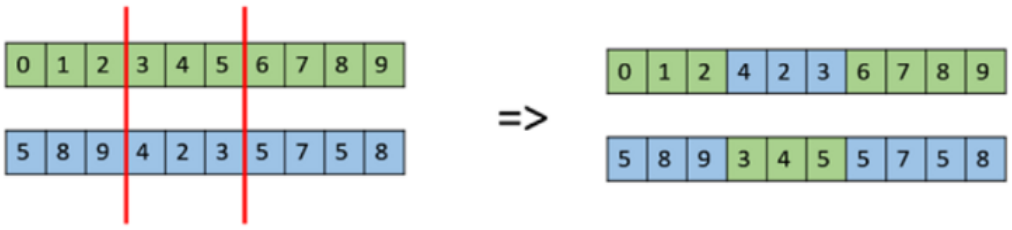

Figure 3.6: Multi-point crossover

3. In uniform crossover, the chromosome is not divided into segments; instead, each gene is considered separately. We are essentially tossing a coin for each chromosome to determine whether or not it will be included in the offspring. We can also associate a coin with one of the parents, giving the child more genetic material from that parent. See Figure 3.7

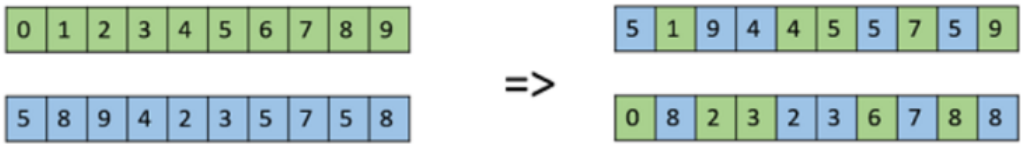

Figure 3.7: Uniform crossover

4. Arithmetic crossover, this is commonly used for integer representations and works by taking the weighted average of the two parents using the following formulas:

$$Child1 = a.x + (1 - a).y$$

$$Child2 = a.x + (1 - a).y$$

Obviously, if a = 0.5, then both children will be identical, as shown in the Figure 3.8.

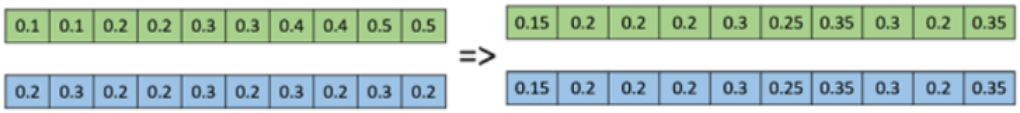

Figure 3.8: Arithmetic crossover.

The choice of crossover operator is dependent on the characteristics of the problem and the representation of the individuals. The objective is to establish a balance between exploration and exploitation, enabling the exploration of new regions of the search space while preserving genetic material from the parents [20].

3.2.4 Mutation

In the genetic algorithm, mutation is the process of randomly changing genetic information in an individual solution (chromosome). It is one of the genetic operations that are applied to a population in the course of evolution.

Mutation plays an important role in the genetic algorithm because it ensures the introduction of new genetic variants into the population. This helps to avoid premature convergence of the algorithm to locally optimal solutions and promotes

diversity breeding in the population [21]. An example of this is shown in Figure 3.9.

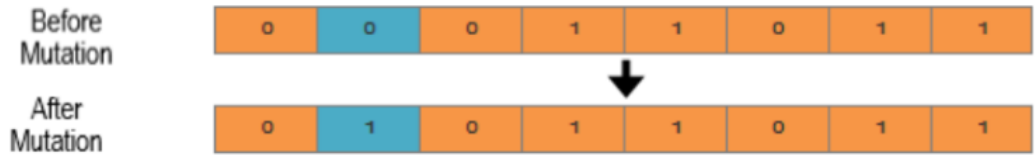

Figure 3.9: Mutation operator.

3.3 Genetic Algorithm Parameters

3.3.1 Population Size

In a genetic algorithm, the number of possible solutions, also called individuals or chromosomes, in each generation is set by the size of the population. It is a very important parameter that affects how exploration and exploitation are balanced in the search field.

With a bigger population, the search area can be explored more, which makes it more likely that better solutions will be found. If there are more kinds of people, the algorithm might find different parts of the search area and not get stuck in local optima. But a bigger population size also means that it takes more time and resources to do calculations and manage the population. With each new generation, it takes more time and memory to evaluate and change a bigger number of individuals.

The best size of a population relies on how hard the problem is and how much computing power is available. For a problem with a bigger search space or a higher level of complexity, a larger population size may help to make sure that the problem is fully explored. But if there aren't enough computers, it might be better to have a smaller population to make things easier on the computers.

Choosing the right population size is often a trade-off between exploration and exploitation, and it relies on the problem at hand. It is normal to try out different

population sizes and see how they affect the algorithm's performance, such as how fast it converges, how good the solution is, and how efficiently it works. Researchers and practitioners can find a good mix between effective exploration and efficient use of available computing resources by carefully adjusting the size of the population. [22].

3.3.2 Crossover and Mutation Rates

In genetic algorithms, crossover and mutation are two important operators that drive the search process and promote diversity in the population. The crossover rate and the mutation rate are two factors that show how likely it is that these operators will be used as the population evolves.

Crossover Rate: The crossover rate shows how likely it is that two parent people will cross over and have children. It shows how the search process is a mix between exploring and using what you find. A higher crossover rate makes it more likely for parents to share genetic information, which leads to a more thorough search of the search area. But if the crossover rate is set too high, it could lead to early convergence or a loss of variety. On the other hand, a low crossover rate can make convergence slow or make it hard to explore different parts of the search area. The best crossover rate relies on how the problem is set up and should be tuned by trying different things.

2. Mutation Rate: The mutation rate tells how likely it is that a mutation operator will be used on an individual, which changes its genetic information in a random way. Mutation helps keep genetic diversity going and can be a way to get out of local optima. By adding random changes, a higher mutation rate makes it more likely that new parts of the search area will be explored. But a mutation rate that is too high can cause too many random changes and stop the process of convergence. On the other hand, if there aren't many changes, there might not be much research. Like the crossover rate, the mutation rate needs to be carefully chosen based on the situation and fine-tuned through experimentation.

People often try out different crossover and mutation rates to find the best ones for a particular situation. The right rates may depend on the nature of the

problem, such as how hard it is to solve, how big the search space is, and how much research and how much exploitation you want to do. Also, the crossover and mutation rates often go hand in hand, and finding the right mix of rates can have a big effect on how well the algorithm works and how good the answers are. [22].

3.3.3 Termination Criteria

In a genetic algorithm, the termination factors tell the algorithm when to stop iterating and give the final answer. These criteria are necessary to make sure the algorithm doesn't keep running forever and gets to a good answer in a reasonable amount of time. In genetic algorithms, there are a few popular ways to end the process:

Maximum Number of Generations: The program stops after a set number of generations. This criterion is helpful when it is expected that the algorithm will find a good answer after a certain number of steps. It sets a fixed limit on the number of generations to keep the computer from doing too much work.

2. Convergence: When a certain amount of convergence is reached, the algorithm stops. You can measure convergence by keeping track of how the fitness value of the best answer changes over time. The algorithm is said to have converged if the improvement goes below a predetermined threshold or stays the same for a certain number of generations.

Fitness Threshold: The algorithm stops when an answer is found that has a fitness value that is greater than a certain threshold. When the goal is to find a solution that meets a certain fitness standard, this criterion is used. Once an answer goes beyond the threshold, it may not be necessary to try again.

4. Time limit: The method stops after a certain amount of time has passed, which has already been set. This factor is especially helpful when there are limits on the amount of computing resources or when results need to be gotten in a certain amount of time.

User-Defined Stopping Condition: The algorithm can be stopped by a user-

defined stopping condition that is relevant to the problem being solved. This condition can be based on information specific to the domain, constraints specific to the problem, or outside factors.

It is important to choose the right criteria for ending a project based on the nature, complexity, and resources of the situation. The criteria for when to stop should strike a mix between giving the algorithm enough time to converge and stopping it from doing too much work. Also, it's a good idea to keep an eye on the algorithm's progress and performance while it's running to make sure that the criteria for stopping capture the desired conditions.

3.3.4 Physical Fitness Evaluation

Physical fitness evaluation, in the context of a genetic algorithm, refers to the process of assessing the fitness or suitability of individuals (chromosomes) in a population based on their ability to solve the problem at hand. This evaluation is crucial for determining which individuals are selected for reproduction and guiding the evolutionary process toward optimal or near-optimal solutions.

In a genetic algorithm for assessing physical fitness, the following stages are typically included:

Each member of the population is represented by a set of genes or variables that encode a possible solution to the problem. These genes may represent numerous aspects of physical fitness, including exercise regimens, dietary regimens, and lifestyle factors.

Genes are decoded or translated into their corresponding phenotypes, which are the actual physical characteristics or traits. This process entails associating specific fitness-related parameters or values with specific genes.

A fitness function is defined as the quantification of an individual's fitness or performance based on their phenotypes. The fitness function measures how well the phenotype of an individual meets the problem's constraints or objectives. It may include a variety of metrics pertaining to physical fitness, such as body composition, cardiovascular health, muscular strength, endurance, and flexibility,

as well as any other pertinent factors.

4. Fitness Evaluation: The fitness function is applied to every member of the population to determine their fitness scores or values. The individual's solution is regarded as more effective the higher the fitness score.

Individuals are selected for reproduction based on their fitness scores, typically using selection methods such as tournament selection or roulette wheel selection. Individuals with higher fitness scores are more likely to be selected, which promotes the propagation of favorable characteristics in the population.

To produce offspring, the selected individuals endure genetic operations such as crossover and mutation. Crossover is the process of combining genetic material from two parents to produce offspring with a mixture of their characteristics. To investigate new regions of the search space, mutation introduces random small changes to genetic material.

7. The stages of fitness evaluation, selection, and reproduction are repeated for multiple generations, allowing the population to evolve toward higher levels of fitness and better solutions.

The genetic algorithm seeks to discover individuals with optimal or near-optimal solutions to the physical fitness problem by repeatedly applying fitness evaluation and genetic operations. Continue until a termination criterion is met, such as reaching a maximum number of generations or attaining an acceptable level of fitness.

Importantly, the fitness evaluation in a genetic algorithm for physical fitness can vary depending on the specific problem being addressed and the desired fitness objectives. The fitness function and evaluation criteria must be meticulously devised to capture the pertinent aspects of physical fitness and to align with the algorithm's objectives.

3.3.5 Elitism

Elitism is a technique utilized frequently in genetic algorithms to preserve the finest individuals or solutions from generation to generation. It ensures that the

finest individuals are passed on to future generations, allowing them to retain their advantageous characteristics and preventing the loss of valuable genetic material.

In a genetic algorithm with elitism, the finest members of the current population are copied directly to the next generation without undergoing any genetic operations such as crossover or mutation. These elite people, also known as the "elite pool," constitute a limited portion of the population that is preserved and protected.

The incorporation of elitism into a genetic algorithm provides numerous advantages:

Preserving Promising Solutions: Elitism ensures that the most promising solutions discovered to date are not lost or diminished. The algorithm maintains a level of progress and prevents the loss of valuable genetic material by passing on the finest individuals to the subsequent generation.

2. Rapid Convergence Elitism can accelerate the algorithm's convergence to optimal or near-optimal solutions. By preserving the best individuals, which are already closer to the optimal solution, the algorithm is able to concentrate its exploration and exploitation efforts on further enhancing those solutions.

Exploitation of Local Optima Elitism assists the algorithm in escaping local optima. Local optima are solutions that appear optimal in a particular region of the search space, but are not optimal globally. By preserving the best solutions, even if they are trapped in local optima, the algorithm has a greater chance of investigating other regions and discovering potentially superior solutions.

The incorporation of elitism into a genetic algorithm is relatively simple. The best individuals in the current population are identified based on their fitness scores following the selection procedure. These individuals are then replicated directly into the next generation, thereby populating a portion of the new population.

The elite pool is typically limited relative to the total population in order to preserve diversity and permit the exploration of new solutions. The remainder of the new generation is produced by genetic operations such as crossover and mu-

tation, assuring the introduction of new genetic material and promoting diversity.

Elitism can be a useful strategy for improving the efficacy and convergence of a genetic algorithm, particularly in the case of complex optimization problems. To avoid premature convergence towards suboptimal solutions, it is essential to establish a balance between preserving the best individuals and encouraging diversity.

Chapter 4

Problem Formulation

4.1 Problem Definition

There are various advantages to using genetic algorithms that make them popular and effective in addressing complex optimization issues. The following are some of the primary advantages of genetic algorithms:

1. Global Search: Genetic algorithms can search a huge search space for globally optimal or near-optimal answers. They are not confined to discovering local optima, but can seek for solutions throughout the entire problem space.

2. Robustness: Genetic algorithms are resistant to noise and changes in the problem space. They are capable of dealing with challenges with noisy or partial data, as well as adapting to changes in the environment or problem limitations.

3. Parallelism: Genetic algorithms are easily parallelizable, allowing for the evaluation of several candidate solutions at the same time. This parallel processing capabilities speeds up and improves the efficiency of the search process.

4. No Derivative Information: Because genetic algorithms do not require derivative information of the objective function or restrictions, they can be used to solve a wide range of optimization problems when getting derivatives is difficult or impracticable.

5. Flexibility: Genetic algorithms are adaptable to both continuous and discrete

variables. They can solve problems with mixed variable types and handle diverse constraints like as equality, inequality, and combinatorial restrictions.

6. Solution Diversity: By maintaining a diversified population of candidate solutions, genetic algorithms encourage solution diversity. This diversity prevents premature convergence to inferior answers and allows for exploration of various sections of the search space.

7. Problem Independence: Genetic algorithms are problem-independent, which means they may be applied to a wide variety of optimization problems without the need for problem-specific adaptations or adjustments. As a result, they are a versatile tool in a variety of areas and sectors.

8. Iterative Improvement: Genetic algorithms operate on the premise of iteratively enhancing the population through the use of genetic operators such as selection, crossover, and mutation. This iterative improvement process refines solutions through generations, eventually leading to better solutions.

Overall, genetic algorithms are well-suited to solve complicated optimization issues where standard optimization techniques may struggle or fail to find appropriate answers.

4.2 Exam Scheduling Constraints

To ensure a feasible and practical exam schedule, a number of constraints must be taken into account. These constraints define the requirements and limitations that must be met during the scheduling procedure. Hard constraints and soft constraints are the two basic categories of constraints[21].

4.2.1 Hard Constraints

Several hard constraints must be satisfied in the exam scheduling problem to assure the feasibility and validity of the generated schedules. Hard constraints are non-negotiable limitations and prerequisites that must be met. The exam scheduling process must adhere to these constraints to ensure impartiality, efficiency, and

practicability.

4.2.2 Soft Constraints

In the exam scheduling problem, while hard constraints must be rigorously satisfied, soft constraints represent desirable conditions and preferences. Soft constraints are not required, but they contribute to the overall quality and schedule optimization. These constraints offer flexibility and trade-offs to improve particular aspects of the solution. The following are examples of gentle constraints encountered in the exam scheduling problem:

The soft constraints allow for trade-offs and adjustments to balance various objectives and preferences while still achieving a satisfactory solution. The optimization process aims to find a feasible schedule that satisfies the hard constraints while optimizing the soft constraints to improve the overall quality of the schedules.

For my implementations, I chose the constraints that are most suitable for Kazakh universities, and for SDU in particular. See Table 4.2.2

4.3 Encoding Scheme

In the context of the exam scheduling problem, the encoding scheme refers to the representation of the exam schedules within the Genetic Algorithm framework. It defines how the solution space is encoded and structured to facilitate the search for optimal or satisfactory solutions. My encoding scheme:

- Each gene represents an exam `\item` command.
- A chromosome represents a complete exam schedule, where each gene is a class that has its own time slot, instructor and department.
- Encoding scheme ensures that no two exams are scheduled in the same time slot, satisfying the time constraints.

Table 4.1: Constraints

Constraints	
1. Hard Constraints	
1.1	Cannot be more than one exam in the same study group at the same time slot.
1.2	One classroom cannot be used for more than one exam at the same time slot.
1.3	One teacher cannot conduct more than one exam in the same time slot.
1.4	The number of seats in the classroom must be greater than or equal to the number of students in the group.
2. Soft Constraints	
2.1	The interval between exams in complex subjects is 2 days.
2.2	Students have a maximum of one exam in one day.
2.3	The number of free seats in the auditorium does not exceed 20% of its total capacity.
2.4	Exams are not allowed between 13:00 and 15:00 on Fridays.
2.5	In one study group, the minimum time interval between exams is 2 days.

Chapter 5

Genetic Algorithm Implementation

5.1 Design and Implementation Overview

In this chapter, I describe the design and implementation of the Genetic Algorithm for solving the exam scheduling problem. This section provides a summary of the most significant considerations and decisions taken during the GA implementation development [23]. The diagram in Figure 5.2 illustrates the view flow of genetic algorithms.

5.1.1 Advantages of GA

There are various advantages to using genetic algorithms that make them popular and effective in addressing complex optimization issues. The following are some of the primary advantages of genetic algorithms:

1. Global Search: Genetic algorithms can search a huge search space for globally optimal or near-optimal answers. They are not confined to discovering local optima, but can seek for solutions throughout the entire problem space.

2. Robustness: Genetic algorithms are resistant to noise and changes in the problem space. They are capable of dealing with challenges with noisy or partial

data, as well as adapting to changes in the environment or problem limitations.

3. Parallelism: Genetic algorithms are easily parallelizable, allowing for the evaluation of several candidate solutions at the same time. This parallel processing capabilities speeds up and improves the efficiency of the search process.

4. No Derivative Information: Because genetic algorithms do not require derivative information of the objective function or restrictions, they can be used to solve a wide range of optimization problems when getting derivatives is difficult or impracticable.

5. Flexibility: Genetic algorithms are adaptable to both continuous and discrete variables. They can solve problems with mixed variable types and handle diverse constraints like as equality, inequality, and combinatorial restrictions.

6. Solution Diversity: By maintaining a diversified population of candidate solutions, genetic algorithms encourage solution diversity. This diversity prevents premature convergence to inferior answers and allows for exploration of various sections of the search space.

7. Problem Independence: Genetic algorithms are problem-independent, which means they may be applied to a wide variety of optimization problems without the need for problem-specific adaptations or adjustments. As a result, they are a versatile tool in a variety of areas and sectors.

8. Iterative Improvement: Genetic algorithms operate on the premise of iteratively enhancing the population through the use of genetic operators such as selection, crossover, and mutation. This iterative improvement process refines solutions through generations, eventually leading to better solutions.

Overall, genetic algorithms are well-suited to solve complicated optimization issues where standard optimization techniques may struggle or fail to find appropriate answers.

5.1.2 System Architecture

Python 3 is used as the programming language for the exam scheduling system, and the SQLite3 database is used for data storage and management. Additionally, the system includes the PrettyTables library for generating visually enticing tables.

The system architecture can be broken down into the following components:

The Genetic Algorithm Module is in charge of putting the genetic algorithm-based strategy for creating exam schedules into practice. It contains the classes and methods required for initialization, selection, crossover, mutation, and fitness evaluation.

The system stores and manages data pertaining to courses, rooms, instructors, students, and scheduling constraints using the SQLite3 database. Python's SQLite3 module is used to access and manipulate the database, enabling for efficient data storage and retrieval.

After the genetic algorithm has generated a schedule, the system constructs tables using the PrettyTables library to display the schedule in an aesthetically pleasing format. The tables contain information about the course, instructor, classroom, and time slots.

5.2 Initialization Stage

During the initialization phase of the genetic algorithm, the goal is to generate an initial set of schedules, each of which can be a potential solution for exam planning. In a population, each graph is considered as an individual or a chromosome.

To initialize the schedules, the full schedule is created as an individual one, and each class in it is represented as a gene. Each genome includes the following characteristics:

1. Room: The room where the exam for the class will be conducted. It describes the room's name and seating capacity.

2. Time: The allotted period for the class's examination. It indicates the day and time section .

3. Instructor: The instructor tasked with instructing the class and supervising the exam. It describes the instructor's courses and name.

4. List of Students: This is a roster of all the students enrolled in the course who will be taking the exam. It provides information about the students' name and courses which he takes.

Combining these parameters for each class yields a comprehensive schedule. The initialization procedure generates multiple schedules at random to create the initial population. The population, represented as a list of schedules, is then prepared for the subsequent phases of the genetic algorithm, including selection, crossover, mutation, and fitness evaluation. The initial population serves as the starting point for the algorithm to explore and evolve improved exam scheduling solutions. Figure 5.1 provides a tabular representation of the initial population.

schedule #	fitness	# of conflicts	classes [dept,course,room,instructor
1	0.333	2	MATH,C1,R1,I2,MT2, MATH,C3,R3,I2,MT4, MATH,C3,R3,I1,MT1, MATH,C9,R2,I7,MT1, IS,C2,R2,I4,MT2, IS,C4,
2	0.333	2	MATH,C1,R1,I2,MT2, MATH,C3,R3,I2,MT4, MATH,C3,R3,I1,MT1, MATH,C9,R2,I7,MT1, IS,C2,R2,I4,MT2, IS,C4,
3	0.333	2	MATH,C1,R1,I2,MT2, MATH,C3,R3,I2,MT4, MATH,C3,R3,I1,MT1, MATH,C9,R2,I7,MT1, IS,C2,R2,I4,MT2, IS,C4,
4	0.2	4	MATH,C1,R1,I2,MT2, MATH,C3,R3,I2,MT4, MATH,C3,R3,I1,MT1, MATH,C9,R2,I7,MT1, IS,C2,R2,I4,MT2, IS,C4,
5	0.1	9	MATH,C1,R1,I2,MT2, MATH,C3,R3,I2,MT4, MATH,C3,R3,I1,MT1, MATH,C9,R2,I7,MT1, IS,C2,R2,I4,MT2, IS,C4,
6	0.091	10	MATH,C1,R1,I2,MT2, MATH,C3,R3,I2,MT4, MATH,C3,R3,I1,MT1, MATH,C9,R2,I7,MT1, IS,C2,R1,I4,MT3, IS,C4,
7	0.091	10	MATH,C1,R1,I2,MT2, MATH,C3,R3,I2,MT4, MATH,C3,R3,I1,MT1, MATH,C9,R2,I7,MT1, IS,C2,R1,I4,MT3, IS,C4,
8	0.053	18	MATH,C1,R1,I2,MT2, MATH,C3,R3,I2,MT4, MATH,C3,R3,I1,MT1, MATH,C9,R2,I7,MT1, IS,C2,R1,I4,MT3, IS,C4,
9	0.02	48	MATH,C1,R1,I1,MT4, MATH,C3,R3,I2,MT4, MATH,C3,R3,I1,MT1, MATH,C9,R2,I7,MT1, IS,C2,R1,I4,MT3, IS,C4,

Figure 5.1: Initial population.

5.3 Selection Stage:

In the selection stage of the genetic algorithm, the goal is to choose the fittest individuals from the current population to proceed to the next generation. The selection process plays a crucial role in driving the evolution of the population towards better solutions to the exam scheduling problem.

One commonly used method for selection is the tournament selection method. In this method, a fixed number of individuals, typically referred to as tournament

size, are randomly selected from the population. The individuals compete against each other, and the fittest individuals are chosen as winners to be selected for reproduction.

By using the tournament selection method, the genetic algorithm ensures that individuals with higher fitness values have a greater chance of being selected for the next generation. This selection process promotes the survival of the fittest individuals and helps propagate their favorable characteristics to future generations. A pseudo code of tournament selection is provided in Algorithm 1.

Algorithm 1 select tournamen

```
1: function _SELECT_TOURNAMENT_POPULATION(population)
2:     tournament_pop ← new Population()
3:     i ← 0
4:     while i < TOURNAMENT_SELECTION_SIZE do
5:         random_index ← random(from 0, to POPULATION_SIZE)
6:         tournament_pop.add_to_schedule(random_schedule from given population)
7:         i ← i + 1
8:     end while
9:     tournament_pop.get_schedules().sort(by fitness)
10:    return tournament_pop
11: end function
```

5.4 Crossover Stage

The crossover stage involves combining genetic information from selected parent individuals to create new offspring. The crossover operator exchanges genetic material between parents at certain points, generating diverse offspring that inherit characteristics from both parents. For the crossover, the best chromosome, or the elite schedule, is preserved without any changes. Then select two parent schedules (chromosomes) using a selection method and cross them by random sampling of genes (classes) from each chromosome. And repeat crossover until the desired number of offspring schedules is generated. Finally Return the offspring population containing the elite schedule and the newly generated offspring schedules. A pseudocode of Crossover is provided in Algorithm 2.

Algorithm 2 Crossover

```
1: function CROSSOVER(population)
2:     elite_schedule ← population.get_best_schedule()
3:     offspring_population ← Population(0)
4:     offspring_population.add_schedule(elite_schedule)
5:     for i = 1 to POPULATION_SIZE − 1 do
6:         parent1 ← select_parent(population)
7:         parent2 ← select_parent(population)
8:         offspring ← create_offspring(parent1, parent2)
9:         offspring_population.add_schedule(offspring)
10:    end for
11:    return offspring_population
12: end function
```

5.5 Mutation Stage

In the genetic algorithm, the mutation stage is responsible for introducing random changes or modifications to the offspring created during the crossover stage. Mutation is a crucial operator that helps maintain genetic diversity in the population and prevents premature convergence to suboptimal solutions.

The mutation process involves randomly altering the genes (classes) of an individual schedule to explore new areas of the search space. This random modification introduces new possibilities and can potentially improve the overall fitness of the population.

For the mutation, the best chromosome, or the elite schedule, is preserved without any changes. Then randomly select one chromosome and mutate them by random changing of gene (class) to new gene (class).

5.6 Fitness Evaluation

The fitness function evaluates the quality of individual solutions or exam schedules based on the defined objectives and constraints. It assigns a fitness value to each individual, indicating how well it satisfies the given constraints and optimization goals. In my implementation, fitness determines the amount of conflict each individual has. My algorithm checks each individual for conflicts. And he counts

them. Then at the end the individual who has the fewest conflicts is the fittest. The algorithm will stop when a generation without conflicts is obtained. A pseudo code of find fitness function is provided in Algorithm 3.

Algorithm 3 calculate fitness

```
1: function CALCULATE_FITNESS
2:     conflicts ← empty list
3:     cls ← list of classes
4:     for i = 0 to length of classes do
5:         if cls[i].roomCapacity < cls[i].maxNumOfStudents then
6:             NumbOfStudentsConflict is happened
7:         end if
8:         for j = 0 to length of classes do
9:             if j ≥ i then
10:                if cls[i].Time == cls[j].Time and cls[i].getId() != cls[j].getId()
   then
11:                    if cls[i].getRoom() == cls[j].get_room() then
12:                        Room Bookings Conflict happen
13:                    end if
14:                    if cls[i].getInstructor() == cls[j].getInstructor() then
15:                        Instructor Booking Conflict happen
16:                    end if
17:                end if
18:            end if
19:        end for
20:    end for
21:    students ← list of students
22:    for i = 0 to length of students do
23:        find_student_time_conflicts(conflicts, students[i])
24:    end for
25:    return 1/(1.0 × (length of conflicts +1))
26: end function
```

5.7 Termination Criteria

In the genetic algorithm, termination criteria determine when the algorithm should stop its execution and declare a solution. These criteria are essential for controlling the optimization process and ensuring that the algorithm does not run indefi-

nitely. In my case, the termination criteria is when the fitness of one chromosome is zero. This means that there are no conflicts in this schedule.

5.8 Algorithm Flowchart

A visual representation of the Genetic Algorithm, an algorithm flowchart is presented in Picture 5.2.

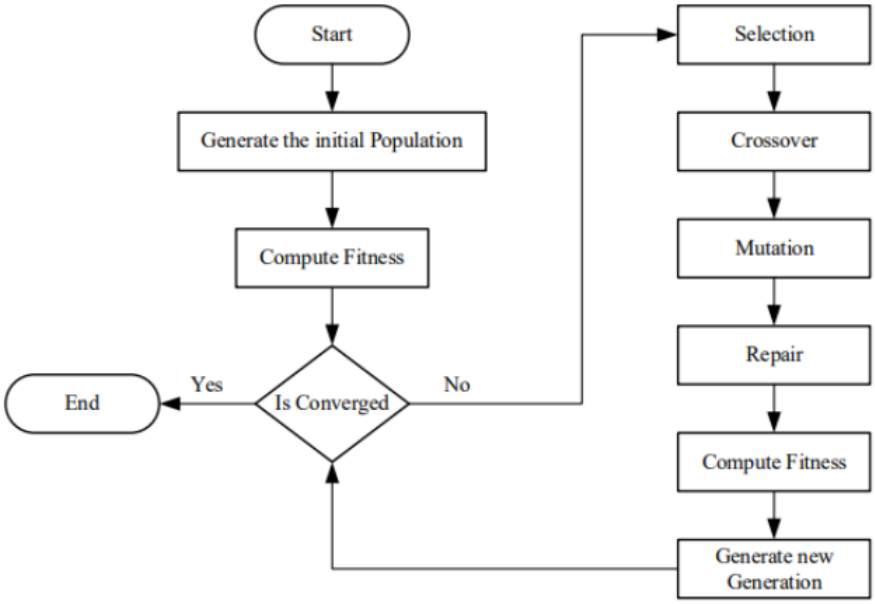

Figure 5.2: The flowchart of genetic algorithms.

Chapter 6

Experimental Evaluation

6.1 Experimental Setup

In this section, I describe the setup and configuration used for conducting the experiments to evaluate the performance of our genetic algorithm for the exam scheduling problem.

6.1.1 Problem Instance

I considered the schedule of an Engineering faculty. Schedule consisted of 9 exams, 50 students, 6 time slots, and 3 available rooms. The instances were generated based on real-world university data, ensuring the relevance and complexity of the problem.

6.1.2 Genetic Algorithm Parameters

Our genetic algorithm was implemented in Python 3. I set the population size to 10, ensuring a diverse set of schedules in each generation. The crossover rate was set to 0.5, and the mutation rate was set to 0.1. I used the tournament selection method with a tournament size of 3 to select parents for reproduction. As a termination criterion, you need to get a schedule where the fitness function is equal to zero. That is, the schedule without conflicts.

6.1.3 Experimental Data

I synthesized data from real university data. Therefore, I can argue that the artificiality of the data will not affect the result in any way. Data includes information on exam schedules, student enrollments, room capacities, and instructor availability. The data was preprocessed to remove any inconsistencies or conflicts and ensure the validity of the problem instances.

6.1.4 Evaluation Metrics

To evaluate the efficacy of our genetic algorithm, I primarily used two metrics: solution quality and computational time. The quality of the solution was determined based on the fitness value of the schedules, which represents the overall compliance of the constraints. In addition, I timed the algorithm's convergence to a solution on a standard desktop computer with an Intel Core i5 processor and 16GB of RAM.

6.1.5 Experimental Procedure

For each instance of the problem, the genetic algorithm was executed ten times with various random seed initializations to account for variance. I recorded the highest fitness value attained by the algorithm, along with the mean and standard deviation for the 10 runs. The algorithm was run until the termination criteria for each run were met.

6.1.6 Hardware and Software Setup

Experiments were performed on a computer with a 3.5GHz Intel Core i5 processor and 16GB of memory. Windows 10 served as the operating system. The genetic algorithm was implemented in Python 3 using the SQLite3 library for database management and the PrettyTables library for table generation and result visualization.

6.2 Dataset Description

In this section, I provide a detailed description of the dataset that was used to evaluate the efficacy of our genetic algorithm for the exam scheduling problem.

6.2.1 Data Source

For data synthesis, data were generated that did not differ in any way from university data. Therefore, I can argue that the artificiality of the data will not affect the results.

6.2.2 Data Preprocessing

Before we used the data set in our tests, it went through a number of steps to make sure the data were accurate and consistent. Among the pre-treatment steps were:

- I cleaned up the data by making sure that each data point represents a different test, student, chamber, or teacher. This got rid of any duplicate or wrong entries.

- Dealing with conflicts: I fixed any problems with the data, such as test schedules that overlapped or rooms that couldn't be used at certain times.

- I have standardized the format of the data set to make sure that information like exam codes, student IDs, room numbers, and teacher names are always shown in the same way.

6.2.3 Dataset Characteristics

The dataset includes the following attributes:

- Exams table: The dataset has a total of N exams, each of which is for a different course or subject given by a university department.Table includes information about name of the course, max number of students and instructors for these courses. See Figure 6.1.

course id	name	max # of students	instructors
C1	408C	20	[I1, I2, I7]
C2	302	25	[I3, I4, I5]
C3	427J	20	[I1, I2]
C4	411	25	[I3, I4, I5]
C5	306	25	[I3, I4, I5]
C6	319K	25	[I3, I4, I5]
C7	312	30	[I3, I4, I5]
C8	303L	30	[I6]
C9	340L	30	[I1, I2, I7]

Figure 6.1: Courses table.

- Student table: The dataset contains data of students, name of students and list of his exams. See Figure 6.2.

student id	name	exams
S001	A. Oregano	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S002	B. Thyme	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S003	C. Basil	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S004	D. Pine	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S005	A. Olive	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S006	F. Lime	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S007	T. peppermint	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S008	W. Vanilla	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S009	I. Basil	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S010	J. Cacao	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S011	M. Walnut	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S012	N. Pecans	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S013	P. Hazelnuts	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S014	K. Cashews	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']
S015	L. Peanuts	['MT1', 'MT2', 'MT3', 'MT4', 'MT5', 'MT6']

Figure 6.2: Students table.

- Room table: The dataset provides information about rooms name and seating capacity. See Figure 6.3.

- Instructor Timetables: The dataset contains information of instructors. Each instructor is associated with their preferred courses. See figure 6.4.

room #	max seating capacity
R1	20
R2	30
R3	25

Figure 6.3: Rooms table.

instructor id	name	courses
I1	Dr. A. Bluebery	['C1', 'C3', 'C9']
I2	Mr. B. Blackbery	['C1', 'C3', 'C9']
I3	Dr. D. Raspbery	['C2', 'C4', 'C5', 'C6', 'C7']
I4	Mr. B. Strawbery	['C2', 'C4', 'C5', 'C6', 'C7']
I5	Dr. D. Bilbery	['C2', 'C4', 'C5', 'C6', 'C7']
I6	Mrs M. Cranbery	['C8']
I7	Dr. A. Whitebery	['C1', 'C9']

Figure 6.4: Instructors table.

6.2.4 Data Representation

The data set is shown in a structured manner that can be used by computers to process it. SQLite tables store the data.

6.3 Experimental Results and Discussion

In this part, I talk about the results of my experiments with the genetic algorithm for the exam scheduling problem. I talk about the success metrics that were found, such as the quality of the schedules that were made and how well the algorithm worked.

6.3.1 Experimental Results

The genetic algorithm effectively generated conflict-free, high-quality schedules. After an average of 192,1 generations, a schedule that satisfied all of the stringent constraints and provided the optimal exam arrangement was discovered. See

Table 6.1: Experimental Results

Exp.	Pop. Size	Mut. Rate	Tourn. Size	Gens	Time (sec)
1	10	0.1	3	87	13
2	10	0.1	3	581	42
3	10	0.1	3	41	4
4	10	0.1	3	244	19
5	10	0.1	3	297	24
6	10	0.1	3	41	4
7	10	0.1	3	67	7
8	10	0.1	3	373	30
9	10	0.1	3	54	5
10	10	0.1	3	136	11
Avg	10	0.1	3	192.1	15.9

Figure 6.5 This result demonstrates that the genetic algorithm is effective at exam scheduling solutions. You can see more details in Table 6.1.

```
+-----+-------------------+-------------------------------------------------+
|  id |    meeting time   | classes [course,room,instructor,meeting-time]   |
+-----+-------------------+-------------------------------------------------+
| MT1 | MON 09:00 - 10:00 |    ['MATH,C1,R1,I7,MT1', 'IS,C5,R2,I4,MT1']    |
| MT2 | MON 10:00 - 11:00 |             ['MATH,C3,R2,I1,MT2']               |
| MT3 | MON 11:00 - 12:00 |     ['IS,C6,R3,I5,MT3', 'CS,C8,R2,I6,MT3']     |
| MT4 | THU 09:00 - 10:00 |   ['MATH,C3,R3,I1,MT4', 'MATH,C9,R2,I7,MT4']    |
| MT5 | THU 10:00 - 11:00 |              ['IS,C2,R3,I3,MT5']                |
| MT6 | THU 11:00 - 12:00 |     ['IS,C4,R3,I4,MT6', 'IS,C7,R2,I3,MT6']     |
+-----+-------------------+-------------------------------------------------+
```

Figure 6.5: Result Schedule.

6.3.2 Computational Efficiency

The experimental results also showed the computational efficiency of the genetic algorithm. On a Core i5 processor with 16 GB of RAM, the execution time for optimal scheduling averaged 15.9 seconds. This indicates that the algorithm is able to solve the exam scheduling problem in a reasonable amount of time, but keep in mind that the scale of the experimental data was rather small.

In general, the results of the experiments show that the genetic algorithm cre-

ates high-quality schedules for the exam planning problem. The algorithm's ability to generate conflict-free schedules and its computational efficiency make it a promising solution for real exam scheduling problems.

Chapter 7

Conclusion and Future Work

7.1 Future Research Directions

1. Experiment with real data on a large scale. I plan to test the performance of the algorithm on 1000 students. Then i will continue with real data from our University

2. Integration of machine learning techniques: Adding machine learning techniques, like reinforcement learning or neural networks, could let the program learn from its past mistakes and change its search strategies on the fly. And increase the performance.

3. Add work with Soft constraints to the algorithm.

4. Combining the genetic algorithm with other metaheuristic algorithms, like simulated annealing or tabu search, could use their complementary powers and possibly lead to better results.

7.2 Conclusion

This work focuses on a special exam scheduling problem that often comes up in universities in Kazakhstan. It suggests a variation of the genetic algorithm as a way to solve the exam scheduling problem.

I go into detail on a real-life test schedule issue for the Faculty of Engineering and Natural Science at SDU. The problem is then solved using a genetic algorithm. We adhere to the rigorous constraints of the Faculty of Engineering scheduling requirements and prioritize them. The proposed model has proven to be effective in terms of creating a conflict-free timetable that allows students to complete their exams in a relaxed environment and allowing the faculty's study and examination office staff to announce an exam schedule that meets the criteria and standards established for the open semester system.

Through this dissertation, I have taken significant strides towards improving the exam scheduling process, and I hope that our work inspires and motivates further research in this field.

Bibliography

[1] Mazin Abed Mohammed, M Ghani, Omar Ibrahim Obaid, Salama A Mostafa, Mohd Sharifuddin Ahmad, Dheyaa Ahmed Ibrahim, and MA Burhanuddin. A review of genetic algorithm application in examination timetabling problem. Journal of Engineering and Applied Sciences, 12(20):5166–5181, 2017.

[2] Mazin Abed Mohammed, M Ghani, Omar Ibrahim Obaid, Salama A Mostafa, Mohd Sharifuddin Ahmad, Dheyaa Ahmed Ibrahim, and MA Burhanuddin. A review of genetic algorithm application in examination timetabling problem. Journal of Engineering and Applied Sciences, 12(20):5166–5181, 2017.

[3] Nelishia Pillay and Wolfgang Banzhaf. An informed genetic algorithm for the examination timetabling problem. Applied Soft Computing, 10(2):457–467, 2010.

[4] Szilvia Jáhn-Erdös and Bence Kövári. Exploring the potential of a genetic algorithm on a real-world complex scheduling problem. In 2022 9th International Conference on Soft Computing & Machine Intelligence (ISCMI), pages 113–117. IEEE, 2022.

[5] Ayhan Ergül. Ga-based examination scheduling experience at middle east technical university. In Edmund Burke and Peter Ross, editors, Practice and Theory of Automated Timetabling, pages 212–226, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-70682-3.

[6] Erik De Corte, Lieven Verschaffel, and Wim Van Dooren. Heuristics and problem solving. In Norbert M. Seel, editor, Encyclopedia of the Sciences of Learning. Springer, Boston, MA, 2012. doi: 10.1007/978-1-4419-1428-6_420.

[7] Kuanyshbek Mamay. Optimizing exam schedules: A literature survey.

[8] Sunith Bandaru and Kalyanmoy Deb. Metaheuristic techniques. Decision sciences, pages 693–750, 2016.

[9] S.S. Rawat and L. Rajamani. A timetable prediction for technical education system using genetic algorithm. Journal of Theoretical and Applied Information Technology, 13(1):59–64, 2010.

[10] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Genetic algorithms and highly constrained problems: The time-table case. In Parallel Problem Solving from Nature, volume 496, pages 55–59, 1991.

[11] E. K. Burke and J. P. Newall. Genetic algorithms applied to university exam timetabling. In Proceedings of the 2003 Congress on Evolutionary Computation (CEC'03), volume 4, pages 2704–2711. IEEE, 2003.

[12] Daniel Kobler. Evolutionary algorithms in combinatorial optimization. In Christodoulos Floudas and Panos Pardalos, editors, Encyclopedia of Optimization. Springer, Boston, MA, 2008. doi: 10.1007/978-0-387-74759-0_167.

[13] A. Doğan and A. Yurtsal. Developing a decision support system for exam scheduling problem using genetic algorithm. Eskişehir Technical University Journal of Science and Technology A-Applied Sciences and Engineering, 22 (3):274–289, 2021.

[14] Ahmad B. Hassanat, V. B. Surya Prasath, Mohammad A. Abbadi, Saleh A. Abu-Qdari, and Hossam Faris. An improved genetic algorithm with a new initialization mechanism based on regression techniques. Information, 9(7): 167, 2018. doi: 10.3390/info9070167.

[15] Anupriya Shukla, Hari Mohan Pandey, and Deepti Mehrotra. Comparative review of selection techniques in genetic algorithm. In 2015 international conference on futuristic trends on computational analysis and knowledge management (ABLAZE), pages 515–519. IEEE, 2015.

[16] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic

acceptance. Physica A: Statistical Mechanics and its Applications, 391(6): 2193–2196, 2012.

[17] Tobias Blickle. Tournament selection. Evolutionary computation, 1:181–186, 2000.

[18] John Grefenstette. Rank-based selection. Evolutionary computation, 1:187–194, 2000.

[19] Padmavathi Kora and Priyanka Yadlapalli. Crossover operators in genetic algorithms: A review. International Journal of Computer Applications, 162 (10), 2017.

[20] Anant J Umbarkar and Pranali D Sheth. Crossover operators in genetic algorithms: a review. ICTACT journal on soft computing, 6(1), 2015.

[21] Michael D. Vose. A closer look at mutation in genetic algorithms. Annals of Mathematics and Artificial Intelligence, 10:423–434, 1994.

[22] Debesh Mishra Mukhopadhyay, Maricel O. Balitanas, Abdulaziz Farkhod, Sang H. Jeon, and Debnath Bhattacharyya. Genetic algorithm: A tutorial review. International Journal of Grid and Distributed Computing, 2(3):25–32, 2009.

[23] Son T. Ngo, Jafreezal B. Jaafar, Izzat A. Aziz, Giang H. Nguyen, and Anh N. Bui. Genetic algorithm for solving multi-objective optimization in examination timetabling problem. International Journal of Emerging Technologies in Learning (iJET), 16(11):4–24, 2021. doi: 10.3991/ijet.v16i11.21017.