University of Sheffield

A Genetic Algorithm for Scheduling Pre-school Sessions

Sheng Xu

Supervisor: Dr Dawn C. Walker

A report submitted in fulfilment of the requirements
for the degree of MSc in Advanced Computer Science

in the

Department of Computer Science

September 10, 2018

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: ______________________________

Signature: ______________________________

Date: ______________________________

Abstract

Children in the 3-4 age group who meet the requirements can receive 15 hours of free weekly childcare in the UK. Because parents have various requirements, each preschool needs to consider a very large number of factors in setting a timetable, which makes it difficult to set schedules manually. So the aim of this project is to create tools that can be used by preschool staff to make schedules easily. This problem can be reduced to scheduling problems so that meta-heuristic algorithms can be used when solving. Meta-heuristic algorithms have a lot of branches and practices as a very popular solution to optimization problems. In this case, the genetic algorithm was chosen as the preference. The genetic algorithm is inspired by the theory of survival of the fittest in evolutionary theory, taking the population as a starting point, forcing the candidate solution to be optimized in the set direction and finally obtaining the optimal solution.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the United Kingdom, the government provides weekly child nursery service to eligible families. Due to constraints such as the number of nursing staff, limited spots and facilities, assigning a timetable to an institution that provides nursery service can be analogized to the scheduling problem that allocates resources to given time slots with given constraints [15]. In this issue, a certain number of children need to be scheduled at the beginning of each semester. When dealing with such scheduling problems, numbers of conditions and constraints need to be clarified. Specifically, for given children, the weekly childcare time of each child is fixed and needs to be allocated to pre-school's various sessions according to the parents' requirements, these are soft constraints. For pre-school, working days are divided into morning, lunch and afternoon three sessions. In addition, the maximum number of children care in each period and the maximum number of children throughout the pre-school is specified, these are hard constraints.

The Genetic Algorithm (GA) is a computational model that simulates the natural selection and genetic mechanism of Darwinian biological evolution. It is a method of searching for optimal solutions by simulating natural evolutionary processes. GA will be used to obtain the optimal solution to the scheduling problem in this project. A detailed description of the concept of GA will be presented in Chapter 2. The scheduling problem in this project is similar to the sub-problem of the work shop scheduling problem: the open shop scheduling problem. However, this has more challenges. For example, they differ in that only one machine can run at each time interval of the open shop scheduling problem but multiple children can be managed at same time slot in this issue. Another challenge is that the conditions for the general scheduling problem are given, but the number of children on the problem will change at the beginning of each entry point. Meanwhile, the schedule is only arranged for the newly added children, the children who have determined the timetable no longer need to be redistributed.

1.1 Aims and Objectives

The Aim of this dissertation is to use the genetic algorithm to build a model and provide pre-school employees with the application to generate schedules by entering parameters. In achieving this aim, the following primary objectives need to be attained:

1. Analyze and comprehend the latest technology in solving scheduling problems using evolutionary heuristics.

2. Investigate what genotype to use can correspond to the phenotype of the result. What format of chromosome can be used as an expression of individuals in GA format

3. Implementation of genetic algorithm construction model to solve problem.

4. Develop an easy-to-use user interface that preschool workers can use to assign sessions to new children.

1.2 Overview of the Report

The remainder of this report is organized as follows: Chapter 2 will investigate existing literature in the area of GA and scheduling issues. Chapter 3 will discuss the demands and analysis of the models that need to be constructed in this paper. Chapter 4 will design the structure of the system, and then complete the code implementation according to the design in Chapter 5. Chapter 6 will analyze, evaluate, and discuss the results. Finally, the entire project is summarized in Chapter 7.

Chapter 2

Literature Survey

This chapter will firstly introduce several frequently used heuristic algorithms and their advantages and disadvantages and application fields. The following content will focus on the concept of genetic algorithm. Then, the advantages and disadvantages of genetic algorithms compared with other algorithms will be discussed. Finally, several papers on applying genetic algorithms to solve problems will be contrasted in detail. Furthermore, their opinions and methods will be evaluated and summarized.

2.1 Heuristic and Metaheuristic

Heuristics are used as new approaches when traditional methods(Exact Approaches) fail to solve problems or solutions are too slow such as NP-Hard problems. Here is a briefly description of P, NP, and NP-hard. The P problem is a problem that can be solved in polynomial time. The NP problem is the ability to verify the correctness of the answer in polynomial time. In the complexity theory, it is indicated that turning an instance of problem A into an instance of problem B, and then indirectly solving problem A by solving problem B, we think that B is more difficult than A. Then, the problem L is said to be NP-hard, if any NP problem can be reduced to a polynomial specification to L [13]. Heuristics is a general term for tactics that can give a feasible solution in an acceptable time and space when solving a specific problem [18]. However, it cannot guarantee optimality of results and unable to determine the deviation of the feasible solution from the optimal solution. Heuristic pursues speed by sacrificing the optimality, completeness, accuracy, and precision of the results. Heuristic algorithms are problem-oriented, many heuristic algorithms are very specialised and depend on the specific problem. Heuristic strategies can change their search path based on personal or global experience in the search for the best solution. When the optimal solution to the problem becomes impossible or difficult to complete (such as NP-complete problem), this strategy is an effective way to obtain a feasible solution [3]. On the contrary, metaheuristic algorithms are problem-independent. They usually do not rely on the special conditions of certain problems, so that they can be applied to a wider range of aspects. Metaheuristic strategies usually impose some requirements on the search process, and then heuristic algorithms implemented according to these requirements are called metaheuristic algorithms [1]. Some of the more well-known metaheuristic algorithms: Simulated Annealing Algorithm (SA), Particle Swarm Optimization (PSO), Ant Colony Algorithm (ACA) and Genetic Algorithm (GA).

2.1.1 Simulated Anneal Algorithm

Simulated Anneal (SA) is a general probabilistic algorithm that finds the best answer to the question in a large search space. Simulated annealing is developed and applied based on the similarities between the annealing process of solids in the physics and general combinatorial optimization problems. The solid is warmed to a sufficiently high temperature, then allowed to cool slowly. As the temperature rises and the internal energy increases, the internal particles of the solid become unordered and the particles gradually become more and more ordered at the time of cooling, at each temperature. After reaching the equilibrium state, the ground state is reached at room temperature and the internal energy is minimized [11]. The generation and acceptance of new solutions to the simulated annealing algorithm can be divided into the following four steps:

- 1. Generate a new solution in the solution space from the current solution by a production function;
- 2. Calculate the objective function difference corresponding to the new solution;
- 3. To judge whether the new solution is accepted, the basis for the judgment is an acceptance criterion;
- 4. When the new solution is determined to be accepted, use the new solution instead of the current solution to begin the next round of testing. Otherwise, the next round of trials will be started without changing the current solution.

The advantage of SA is that it can handle objective functions with any degree of non-linearity, discontinuity, and randomness, and the objective function can also have arbitrary boundary conditions and constraints. Compared with other linear optimization methods, this algorithm has less programming workload and ensures that the global optimal solution can be found from a statistical point of view. The disadvantage is that compared to other technologies, it is more difficult to modify parameters when solving specific problems. Improper use can cause it to become simulated quenching, leading to the inability to statistically find a global optimal solution [5].

2.1.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an evolutionary algorithm (EA). Similar to the simulated annealing algorithm, it also starts from a random solution and iteratively finds the optimal solution [19]. The fitness is used as the criterion for evaluating the quality of the solution. It seeks the global optimum by following the current searched optimal value.

Specifically, in the PSO, the solution to each optimization problem is a bird which called particles in the search space. All particles have a fitness value that is determined by the function being optimized. Each particle also has a speed that determines the direction and distance that they fly. The particles then follow the current optimal particle search in the solution space [16]. The standard PSO algorithm flow is as follows:

- 1. Initialize a group of particles, including random positions and speeds;
- 2. Evaluate the degree of fitness of each particle;
- 3. For each particle, compare its fitness value with the individual optimal solution it has experienced, and if it is better, use it as the current individual optimal solution; for each particle, its fitness value and Compare the global optimal solution experienced, if it is better, reset the index number of the global optimal solution;
- 4. Change the speed and position of the particles according to the equation;
- 5. If the fitness value meets the requirement or reaches the preset value, it ends; otherwise, it returns to step 2.

The advantage of the particle swarm algorithm is that compared with the genetic algorithm, the rules are set up without crossover and mutation operations, and the implementation is easy, the precision of the result is high, and the convergence is fast. However, because the population is initialized randomly, the result of the final iteration is greatly affected by it, and the global optimal solution may not be found due to convergence too fast [19, 16].

2.1.3 Ant Colony Algorithm

Ant Colony Algorithm (ACA) is an algorithm based on probability and path search It was inspired by the behavior of ants in finding the path during the search for food. In the process of ant foraging, the behavior of a single ant is relatively simple, but the ant group as a whole can reflect some intelligent behavior [17]. For example, ant colonies can find the shortest route to a food source in different environments. This is because the ant within the ant group can transmit information through some kind of information mechanism. In fact, the ant releases a substance that can be called pheromone on its path. Ants within the ant colonies have the ability to sense pheromones, they walk along a higher concentration of pheromones, and each passing ant leaves pheromones on the road. This creates a mechanism similar to positive feedback. After a period of time, the entire ant colony will reach the food source along the shortest path [12]. When using the ant colony algorithm to solve a specific problem, the following basic ideas are needed: abstract the various feasible solutions of the problem into the walking path of the ant colony, so all the paths of the entire ant colony constitute a set of all solutions of the whole problem. Ants leave pheromones on the path, and shorter paths have more pheromones. As the process of the solution continues to deepen, the concentration of pheromone accumulated on the shorter path gradually increases, and the number of ants in the selected path increases. Finally, under the positive feedback, the best path of the entire ant colony will be found. The corresponding solution for this path is the optimal solution to the problem. [17].

Compared with other heuristic algorithms, ant colony algorithm has strong robustness in solving performance (a slight modification to the basic ant colony algorithm model and can be applied to other problems) and the ability to search for a better solution. Ant colony algorithm can be easily combined with multiple heuristic algorithms to improve algorithm performance [9, 4]. It adopts a positive feedback mechanism, which makes the search process converge and eventually approaches the optimal solution. The search process of this algorithm uses a distributed computing approach, where multiple individuals perform parallel computations at the same time, greatly improving the computational power and operational efficiency of the algorithm [9].

2.2 Genetic Algorithm

The genetic algorithm is also a kind of metaheuristic algorithm. Since this algorithm will be used in this project, this section will introduce the genetic algorithm in more detail than other metaheuristic algorithms. The genetic algorithm is also an evolutionary algorithm. It was first proposed by Professor H. Holland of the University of Michigan, and originated from the study of natural and artificial adaptive systems in the 1960s [10]. It was inspired by Darwin's theory of evolution. It is a method of simulating the natural evolution process to search for optimal solutions. It models the biological evolution of Darwin's biological evolution theory by simulating the natural evolution and genetic mechanism of biological evolution. There are many issues that apply to genetic algorithms, such as pathfinding problems, 8 digital problems, prisoner dilemmas, motion control, finding the center of the circle problem (in an irregular polygon, looking for the center of a circle that contains the largest circle in the polygon), TSP Problems, production scheduling problems, artificial life simulation, etc.
There are many terms in the genetic algorithm that need to be clarified:

- **Genotype**: the internal representation of the trait chromosome;
- **Phenotype**: The external manifestation of a trait determined by a chromosome, or the external appearance of an individual formed from a genotype;
- **Evolution**: The population gradually adapts to the living environment and its quality is continuously improved. The evolution of biology takes place in the form of populations.
- **Fitness**: Measuring the adaptation of a species to its living environment.
- **Selection**: Select a number of individuals from a population with a certain probability. In general, the selection process is a process based on the fitness of the survival of the fittest.

- **Crossover**: DNA is cleaved at one and the same position on two chromosomes, and the two strings are crossed to form two new chromosomes. Also called gene recombination or hybridization;
- **Mutation**: There may be (probably a small probability) some replication errors in replication, and mutations produce new chromosomes that exhibit new traits.
- **Coding**: The genetic information in DNA is arranged in a pattern on a long chain. Genetic coding can be seen as a mapping from phenotypes to genotypes.
- **Decoding**: The mapping of genotypes to phenotypes.
- **Individual**: An entity with a characteristic chromosome;
- **Population**: A collection of individuals whose number is referred to as the size of the population.

The process of genetic algorithm implementation is like the evolution of nature. First, look for a scheme that digitalizes a potential solution to a problem, ie, establish a phenotypic and genotypic mapping relationship. Then initialize a population with random numbers. Individuals in the population are these digital codes. Next, let the population begin to multiply, use crossover ways to generate offspring, use mutations to generate offspring to make more progeny possible, and then use the fitness function for each individual gene after the appropriate decoding process. Make a fitness assessment. Use the selection function to select the required individuals according to the constraint set by the problem, remove individuals that do not meet the requirements, and finally repeat the process of reproduction and selection. The genetic algorithm does not guarantee that the optimal solution to the problem can be obtained. However, the greatest advantage of using the genetic algorithm is that it is not necessary to focus on finding the optimal solution in the process but to delete some individuals who are not performing well.

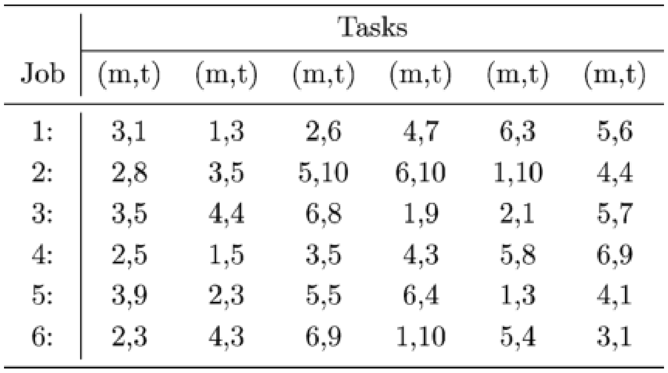
Genetic algorithm has good practicability to problems in different fields. In its search process, the evaluation function is used to inspire, so the process is simple. In addition, genetic algorithms have superior performance in global search compared to other algorithms. This means that the fast trap of falling into the local optimal solution can be avoided in the disintegration process, and the entire solution in the solution space can be quickly searched. In particular, it can leverage its inherent parallelism to implement distributed computing to accelerate the solution. In the modern heuristic algorithm, many problems are solved by the combination of different algorithms, and the genetic algorithm is extensible, which is very suitable for combination with other algorithms.

2.3 Past Researches

This section will review and contrast several existing examples of genetic algorithm applied to scheduling problems

2.3.1 Problem Description

Scheduling is a generic term for a class of planning issues, and this is a derivative problem of the complication of a typical job shop scheduling. Some of the past studies used genetic algorithms to solve similar problems and get ideal results. [7] provided a typical example of a job shop scheduling problem. As shown in the Figure 2.1, there are six tasks and six machines in a work shop. Each job has to use six machines (unordered) but However, each job occupies a different usage time. The problem is how to arrange these six tasks so that the time to complete all jobs is the shortest. This figure is also used as a classic case in [6]. The number of machines and the number of jobs in the figure are not fixed. When the number of jobs is small, such as two jobs, the problem can be easily solved optimally, but when the number of jobs Increased, the problem is NP-Hard. When such problems are added with some constraints and more conditions, more complex problems will arise. It is indicated that the permutation manufacturing-cell flow shop (PMFS) is a sub-problem for the workshop problem. In each workshop there are some tasks and some groups. Each group has some members. When determining the schedule, each group is determined at the same time. The order and working order of the members within the group [2]. Different from [2], the object of [8] is the multi-objective scheduling problem. The traditional scheduling problem ultimately requires the shortest completion time, but in the reality factory, due to the complexity of the work task, it may be necessary to achieve multiple goals at the same time. For example, in the problem description of [8], the following conditions are given: the number of processing machines and storage capacity, the production process, the processing time, the quantity of raw materials, and the minimum requirement of the product. After removing the original resource constraints and process constraints, the ultimate purpose of the problem includes: determine the order and time for each processing unit to perform tasks, set appropriate batches of tasks, allocate raw materials, and minimize the corresponding timetable. Completion time, or maximize total production over a fixed time frame. Although the established issues are different, but these researches all use genetic algorithms to solve the problem, the steps to solve the problem are similar.

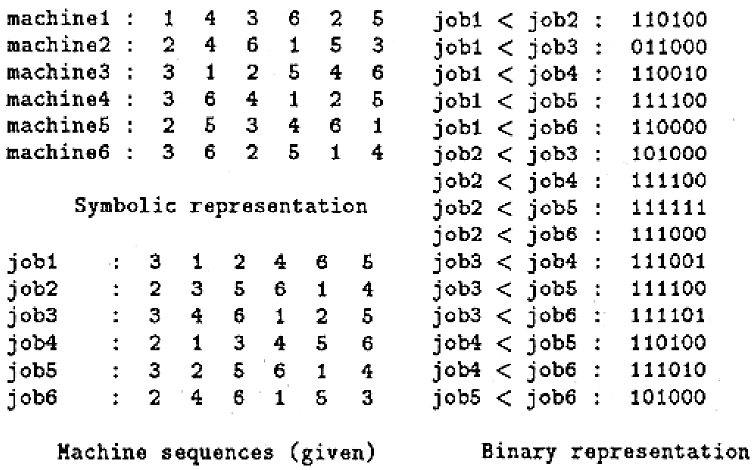


	Tasks					
Job	(m,t)	(m,t)	(m,t)	(m,t)	(m,t)	(m,t)
1:	3,1	1,3	2,6	4,7	6,3	5,6
2:	2,8	3,5	5,10	6,10	1,10	4,4
3:	3,5	4,4	6,8	1,9	2,1	5,7
4:	2,5	1,5	3,5	4,3	5,8	6,9
5:	3,9	2,3	5,5	6,4	1,3	4,1
6:	2,3	4,3	6,9	1,10	5,4	3,1

Figure 2.1: An example of Job-shop scheduling [7]

2.3.2 Chromosome Representation

This section will compare previous studies from the aspects represented by chromosomes. The first challenge when using genetic algorithms is to create a proper representation. In other words, each feature of the result is first coded to form a phenotype for the genotype, and then the codes are combined together to become a chromosome. There are many kinds of encoding of genetic algorithms, such as binary encoding, Gray code encoding, real number encoding, matrix encoding and so on. Binary encoding uses binary coded symbol set 0, 1, and each individual is a binary symbol string. The [10]'s genetic algorithm is a binary coded GA. Among the problems solved by various genetic algorithms, the binary coded algorithm deals with the most patterns. Almost any problem can be expressed in binary code. Therefore, binary coding applications are the earliest and most extensive, and it is the most commonly used coding scheme in genetic algorithms.



machine1 :	1	4	3	6	2	5
machine2 :	2	4	6	1	5	3
machine3 :	3	1	2	5	4	6
machine4 :	3	6	4	1	2	5
machine5 :	2	5	3	4	6	1
machine6 :	3	6	2	5	1	4

Symbolic representation

job1 :	3	1	2	4	6	5
job2 :	2	3	5	6	1	4
job3 :	3	4	6	1	2	5
job4 :	2	1	3	4	5	6
job5 :	3	2	5	6	1	4
job6 :	2	4	6	1	5	3

Machine sequences (given)

job1 < job2 :	110100
job1 < job3 :	011000
job1 < job4 :	110010
job1 < job5 :	111100
job1 < job6 :	110000
job2 < job3 :	101000
job2 < job4 :	111100
job2 < job5 :	111111
job2 < job6 :	111000
job3 < job4 :	111001
job3 < job5 :	111100
job3 < job6 :	111101
job4 < job5 :	110100
job4 < job6 :	111010
job5 < job6 :	101000

Binary representation

Figure 2.2: Representation of 6x6 schedule [14]

For the classic 6x6 job shop scheduling problem, [14] specifically shows the coding in Figure 2.2. It can be clearly seen that [14] first used the encoding of real numbers and then converted them to binary encodings, because binary encoding can make subsequent selection, crossover, and mutation operations easy to implement. [2] proposed two ways to express chromosomes for permutation manufacturing-cell flow shop problems. The first one is shown in Figure 2.3, where f denotes the workshop team (also called family) and J denotes different tasks. Obviously, this coding method divides chromosomes into two segments. The first segment is the sequence of the working group, ie,$f_3 \rightarrow f_2 \rightarrow f1$, and the second segment is the order of specific tasks within the group. For example, the sequence of tasks in f_1 is $J_1 \rightarrow J_3 \rightarrow J_2$. The second representation method is shown in Figure 2.4. This method reduces the length of the first representation. It removes the representation of the first sequence of chromosomes for the family, but adds a new line to represent the corresponding Task-dependent family. Then, from the second row of representations, the order of the family in the entire work can be obtained by the order in which each family first appears, namely, the first paragraph of the first chromosome representation method. This implicit representation and the way to change from a one-line representation to a two-line representation are very instructive.

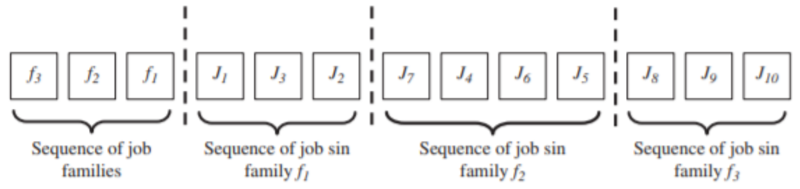


Figure 2.3: One representation of PMFS [8]

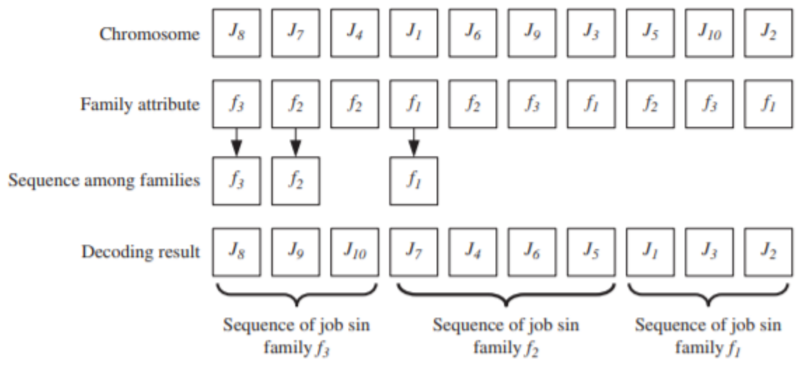

Chromosome
J_8 J_7 J_4 J_1 J_6 J_9 J_3 J_5 J_{10} J_2
Family attribute
f_3 f_2 f_2 f_1 f_2 f_3 f_1 f_2 f_3 f_1
Sequence among families
f_3 f_2 f_1
Decoding result
J_8 J_9 J_{10} J_7 J_4 J_6 J_5 J_1 J_3 J_2
Sequence of job sin family f_3
Sequence of job sin family f_2
Sequence of job sin family f_1

Figure 2.4: Another Representation PMFS [8]

2.3.3 Genetic Operator

When the chromosome representation is determined, the genetic algorithm can be calculated. The first step is usually to initialize the population. Determine the size of the population, and then generate individuals of a defined size as the initial population (feasible solutions) according to the encoding of the chromosomes. The fitness function is determined according to the constraint of the problem, and the objective function is determined according to the target. The objective function only appears when the problem gives a clear goal, such as minimum completion time. When the problem does not have a clear goal but only optimization, the objective function may not be needed. Then start using the operator to operate on the population.

Selection: Use the fitness function and objective function to screen the current population. Individuals that perform well will be cross-operated to allow good genes to be inherited. Individuals who perform poorly will undergo mutations to allow the individual to generate new and better genes to optimize the population.

Crossover: New chromosomes is created by swapping parts of two chromosomes. The probability of crossing needs to be set in advance as a parameter. However, the cross operation needs to determine the specific operation method according to the specific problem. The problem of [2] belongs to the job shop scheduling problem, so the task in the chromosome is ten different tasks. Further, there are ten different tasks that need to be maintained in the chromosomes of the offspring after crossover. So as shown in Figure 2.5, first select an individual of the current population as one of the parents, and then randomly generate a string of 0/1 strings based on the length of the chromosome, and then according to the position of 1 in the Mask (corresponding Task subscripts) Identify the tasks that will be preserved in the parent and arrange them in the order of the first parent's chromosomes. If keeping J_2, J_4, J_5, J_6, J_9, then select another parent chromosome to install the second parent's chromosomes for the remaining tasks In order. This creates a new individual that inherits the two current population chromosomes. Then, when the encoding of the chromosome is binary, this operation becomes much simpler. Two chromosomes of the parent are selected, and then one or more random intersections are shown as 2.6, and the parent's The chromosomes are split from the intersections and recombined into the new two offspring.

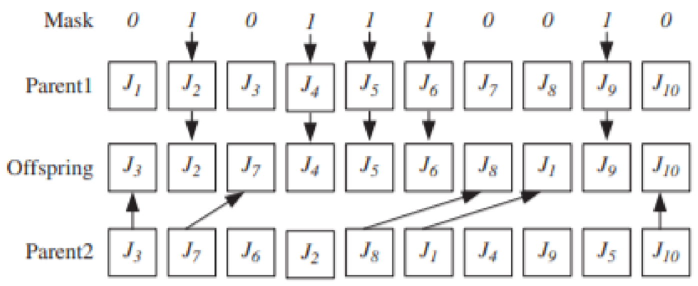

Mask 0 1 0 1 1 1 0 0 1 0
Parent1 J1 J2 J3 J4 J5 J6 J7 J8 J9 J10
Offspring J3 J2 J7 J4 J5 J6 J8 J1 J9 J10
Parent2 J3 J7 J6 J2 J8 J1 J4 J9 J5 J10

Figure 2.5: Crossover Operation of PMFS [8]

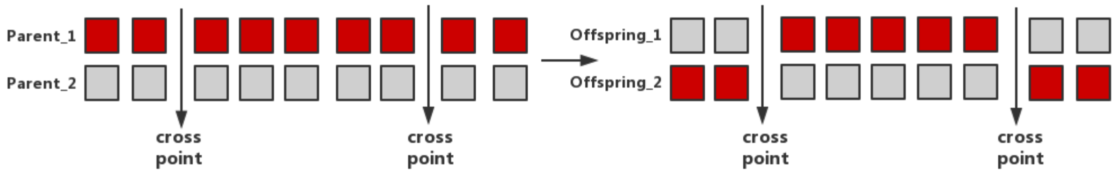


Figure 2.6: Crossover Operation

Mutation: Mutation operations are generally directed to individual individuals. When the chromosome is represented in binary, it is only necessary to randomly convert one or more digits (0 or 1) to another digit (1 or 0). This also mimics the catastrophe theory of biological evolution, so the mutation also needs to set the probability of mutation. When the coding of the chromosomes is not binary, the method needs to be set according to the specific problem. Chen uses two methods, exchange and insertion when the operator is mutating. The exchange is to randomly pick two different locations from the chromosome, and then swap the J at the corresponding location. Insertion is to randomly select a gene from a chromosome and insert it into other locations so that both the position and the subsequent gene move backwards by one. This will ensure that the number of tasks and task subscripts meet the criteria of the chromosome. Obviously, the insertion may cause huge changes in the chromosomes, which may lead individuals to produce better phenotypes.

Every time a round of such operations is completed, a new generation of population will be generated, and the population will be evolving towards the set direction. The more iterations, the more optimal the solution is, but the more time it takes. So the number of iterations is also a parameter that needs to be considered.

2.4 Summary

In the past two decades, meta-heuristic algorithms have been widely used in the face of complex problems in spatial search. These problems cannot be easily solved with precise algorithms because they are mostly NP-hard. Genetic algorithm is an excellent algorithm that can be used to solve the scheduling problem. In the field of improving the performance of genetic algorithms, in the past most people focused on the optimization of operators. [2] believes that the development of new chromosome representations can improve the performance of existing algorithms. In the same vein, Fang in his paper notes that more appropriate mapping of genotypes and phenotypes can reduce the probability of premature convergence of the genetic algorithm. [8] states that while genetic algorithms can be used in various optimization problems as one of the most adaptable algorithms, it is necessary to consider the chromosomal constraints of specific problems during the operation of the operator.

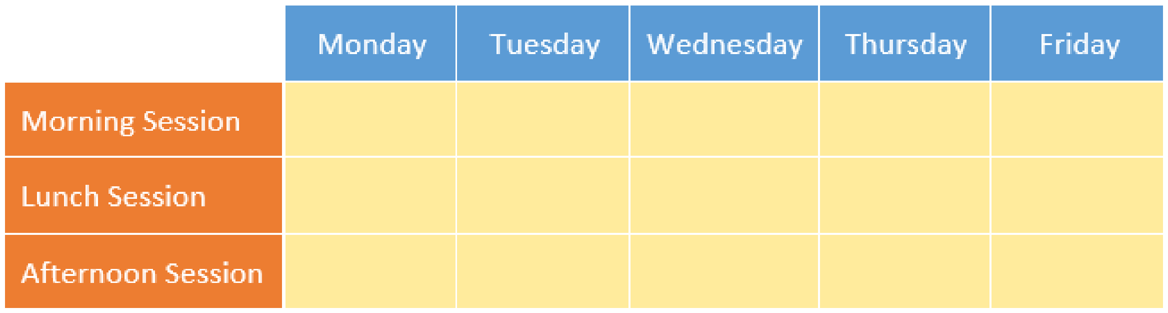
Chapter 3

System Requirement and Analysis

In this chapter, the description of the certain issue will be claimed in detail, the various constraints that the solution algorithm needs to consider will be enumerated. The effects of these constraints on the design of the algorithm are further analyzed. Identify the major challenges in the solution. Finally analyze the risks and possible legal and ethical issues.

3.1 Problem Description

Employees in an institution that provides 3-4 year old child care services face such a difficulty. They need to develop a timetable as shown in 3.1, students who have already registered will be reasonably arranged in each session. In the process of scheduling, they need to consider the total duration requirements of each student, the student capacity of each session and each parent's preference for the session arrangement. The specific description will be filled in section 3.2. Therefore, such an optimization problem will be solved using genetic algorithms. The specific constraints will be met by designing and upgrading operators of GA.



	Monday	Tuesday	Wednesday	Thursday	Friday
Morning Session					
Lunch Session					
Afternoon Session					

Figure 3.1: Schedule Form of Pre-school

3.2 Requirement

In simple terms, the requirement of this project is to provide preschool employees with tools that can automatically generate a child schedule. As the government provides 15 hours free care support for 3-4 year-old children who meet the requirements of the family. Since families have the right to choose different care institutions, care providers cannot determine how many children will join their care services on each school day. Therefore, it is a very complicated task for employees to arrange an appropriate timetable for their children. At the same time, the time and place of the parents work will also affect the time they send and pick up the children, which will further affect the difficulty of setting the timetable for the school. The number of students registered in this preschool program is 52. Each session can have up to 26 students in school at the same time due to staffing and space constraints. Each weekday can be divided into three sessions. The morning session and the afternoon session are 2.5 hours from 8.50-11.20 am and 12.35-3.05 pm respectively. The lunchtime session is 1.25 hours in length between the two sessions from 11.20-12.35 am. Each student can select only the morning session and afternoon session but cannot only select the lunchtime session. They also can choose the first two sessions, the last two sessions or three sessions. So each student can have five different options every day. Each child has 15 hours of free nursing time per week, but not all need to be allocated in the same preschool, and their parents can buy extra hours for their children. In general, children who are three years old must register at the school's school entrance date. The preschool has three terms per year, and the beginning of each term is the point in time for the registration of new students. so there are three entry points as shown in 3.2. Finally, there are priority levels when arranging courses for students. The highest priority is Special Educational students, such as children with physical or mental deficits. The second is the age of the child. The older the child, the higher the priority. The third is the special request of the parents. The afternoon session.

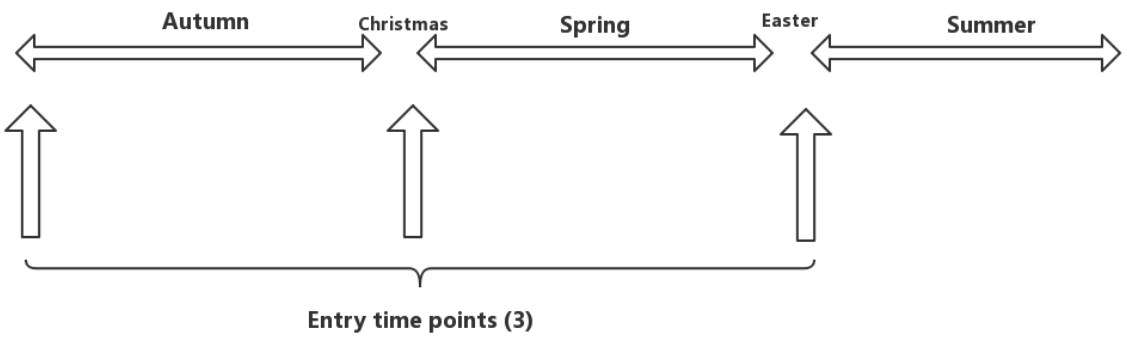


Figure 3.2: Preschool entrance point for each year

Through the understanding and analysis of the problem, the following is an enumeration of the problem constraints. It will be divided into two parts: **hard constraints** and **soft constraints**. The hard constraints represent the conditions that must be met. If it is not satisfied that the hard constraints represent the results, the soft constraints refer to the constraints that are satisfied as much as possible, and the final results can accept that some of the conflicts do not satisfy these constraints.

- **Hard constraints**:
  - H1 Maximum number of students registered simultaneously: 52
  - H2 Maximum number of simultaneous students with session: 26
  - H3 Can't choose lunchtime session alone
  - H4 Student's weekly required attendance time (15 hours free + additional paid time)
  - H5 Schedule only for newly enrolled children on each entry point. The original childs timetable remains unchanged if their parents do not have special requirements.
- **Soft Constraints**:
  - S1 Parental Preferences
  - S2 Older students have higher priority
  - S3 Students with disabilities have the highest priority

3.3 Requirements Analysis

The first step in using genetic algorithms to solve the scheduling problem in this project is still the establishment of a chromosome representation. Binary coding will be the preferred choice for coding. Each chromosome represents an individual and also represents a possible outcome. Since preschool's timetable is counted by each person, it is a total of 15 sessions of three sessions per day, five working days per week, with 1 indicating that the child is in the time period was arranged for care, and 0 was used to represent that the child did not have any care during this time. Then each child's timetable can be expressed as a string of 0s or 1s of length 15. However, because there is a limit to the maximum number of people in each session of the school, the representation of chromosomes cannot be based on the timetable of each child, but rather on the timetable of the entire school. Such a chromosome is not a one-dimensional binary code, but two-dimensional. Each line represents a child's weekly schedule. In this way, each child's weekly total care time and their parents' requirements can be considered in each row. In each column it is easy to calculate the maximum number of students per term and the maximum number of people per session.

3.4 Evaluation and testing

Evaluation and testing is an important part of any software and system development task because its purpose is to assess whether the system meets requirements and test system performance. Unlike other system designs, this project focuses on an optimization problem, and the final result will not necessarily be an optimal solution. Therefore, it is not possible to set a perfect prediction value and then compare the final operation result with the prediction to evaluate the system. However, the performance of system can be measured by the fitness score. The testing part will be divided into the following three parts.

- Unit test - Since the eventual results cannot be predicted and evaluated, unit testing is performed on each piece of code to ensure that the output and expected results of each piece of code are in line with expectations. At the same time, we must also ensure that the code implementation is consistent with the system design.

- Integration test - Randomly generate constraints that conform to the format according to the specific constraint format. Then test whether the system can complete the optimization task under the condition of random constraints. In this way, the robustness of the system is tested.

3.5 Risk Analysis

In order to help developers manage time and resources to successfully complete the entire project, it is necessary to identify in this section the factors that may hinder the completion of this work within the specified time. Likelihood is an indication of the probability of a risk. Impact is a measure of the impact of risk on project success. The integration of likelihood and impact is the level of risk that a certain factor contributes to the completion of a project. It is represented by the following equation 3.1, with levels of 1 to 4 from low to high, indicating levels of likelihood and impact.

$$risk = likelihood \times impact \tag{3.1}$$

3.3 will detail a number of risks and measures to reduce the impact of each risk.

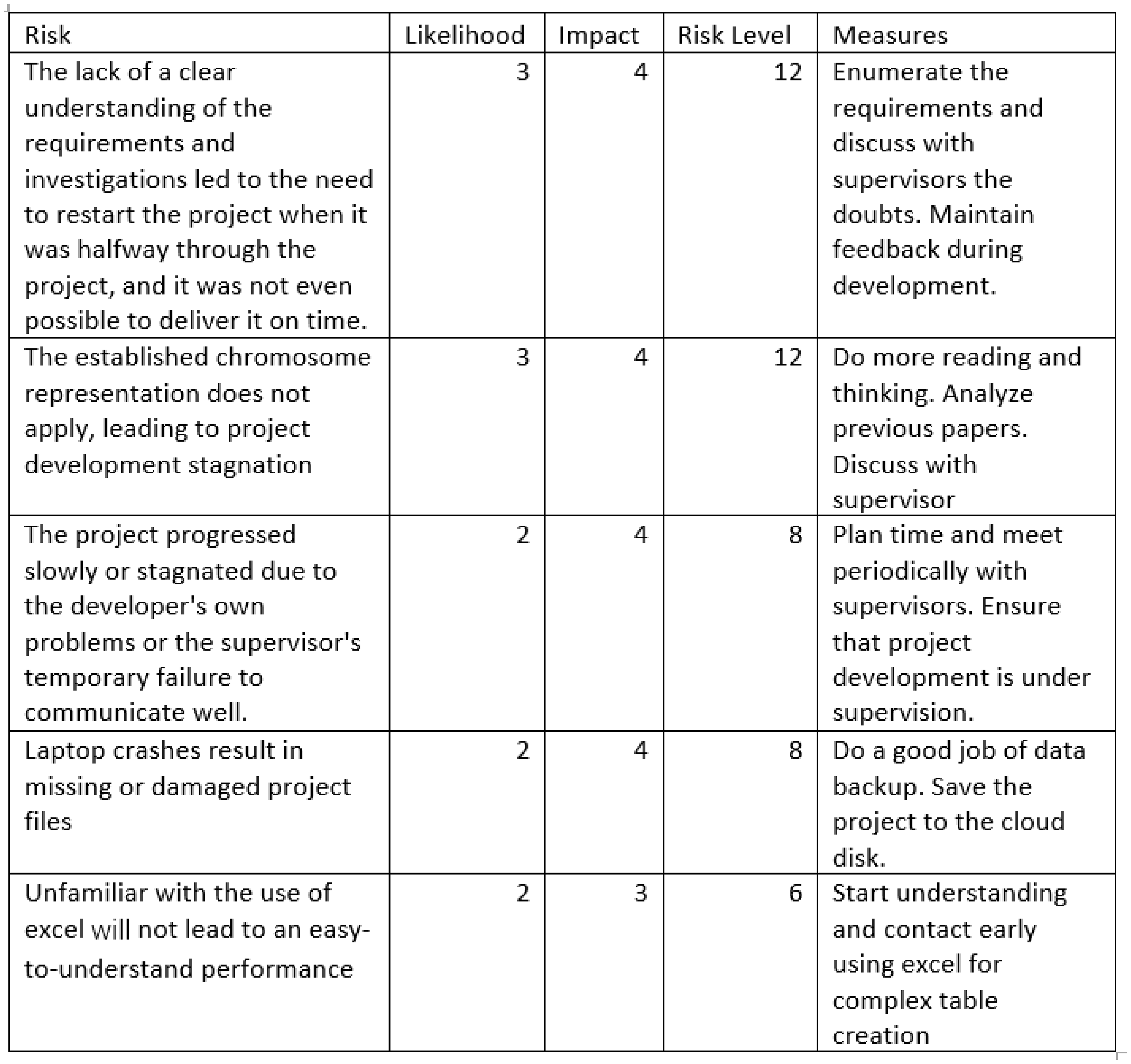
3.6 Ethical, Professional and Legal Issues

At the ethical and legal level, this project will follow the six key principles outlined in the ESRC Framework for Research Ethics. The actual data will not be used during the implementation of the entire project but instead dummy data will be used instead in order to ensure the security of information. In addition, when selecting test volunteers, they are guaranteed to participate voluntarily. The materials mentioned in this dissertation include picture tables that will be referenced if not created by the developer.

3.7 Summary

This chapter outlines the issues that need to be addressed and lists the requirements in terms of hard constraints and soft constraints.

In addition, test methods from three different perspectives will be applied to the design process of the system. Finally, Some of the ethical and legal aspects of the dissertation project is also explained.



Risk	Likelihood	Impact	Risk Level	Measures
The lack of a clear understanding of the requirements and investigations led to the need to restart the project when it was halfway through the project, and it was not even possible to deliver it on time.	3	4	12	Enumerate the requirements and discuss with supervisors the doubts. Maintain feedback during development.
The established chromosome representation does not apply, leading to project development stagnation	3	4	12	Do more reading and thinking. Analyze previous papers. Discuss with supervisor
The project progressed slowly or stagnated due to the developer's own problems or the supervisor's temporary failure to communicate well.	2	4	8	Plan time and meet periodically with supervisors. Ensure that project development is under supervision.
Laptop crashes result in missing or damaged project files	2	4	8	Do a good job of data backup. Save the project to the cloud disk.
Unfamiliar with the use of excel will not lead to an easy-to-understand performance	2	3	6	Start understanding and contact early using excel for complex table creation

Figure 3.3: Risk Register

Chapter 4

System Design

This chapter will present the specific design of the system to solve the target scheduling problem. These designs include input and output, chromosomes, fitness functions, selection operators and crossover operators. The division and flow of the overall design are shown in the figure 4.1.

4.1 Design Methodology

This project will be developed using a spiral model, which means that after each round of implementation, the requirements need to be checked and the completion of the verification results. If there are any objectives that are not achieved, the implementation of the function needs to be increased. If the ideal fitness is not achieved, Need to improve the operator and other calculation parameters.
Under this development model, functions can be separated in a modular manner, reducing coupling. This ensures that individual functions can be tested individually without affecting each other. At the same time, the framework is completed in a short period of time (under the condition of discarding partial constraints). After completing the framework, increase the constraints incrementally, improve the operators, and ultimately improve the applicability of the results.

4.2 Input and Output

Since end-user provides an Excel spreadsheet file, this file provides a list of registered students and some properties. The specific format of this file will be described in detail in Chapter 5. So this file can be used as input. That is, the program can automatically read data from this file and process it for a specific data format that is convenient for subsequent processing. In addition, this file itself contains the format of the timetable, so this file can also be used as an output file, that is, the result of running the program is written to the file.

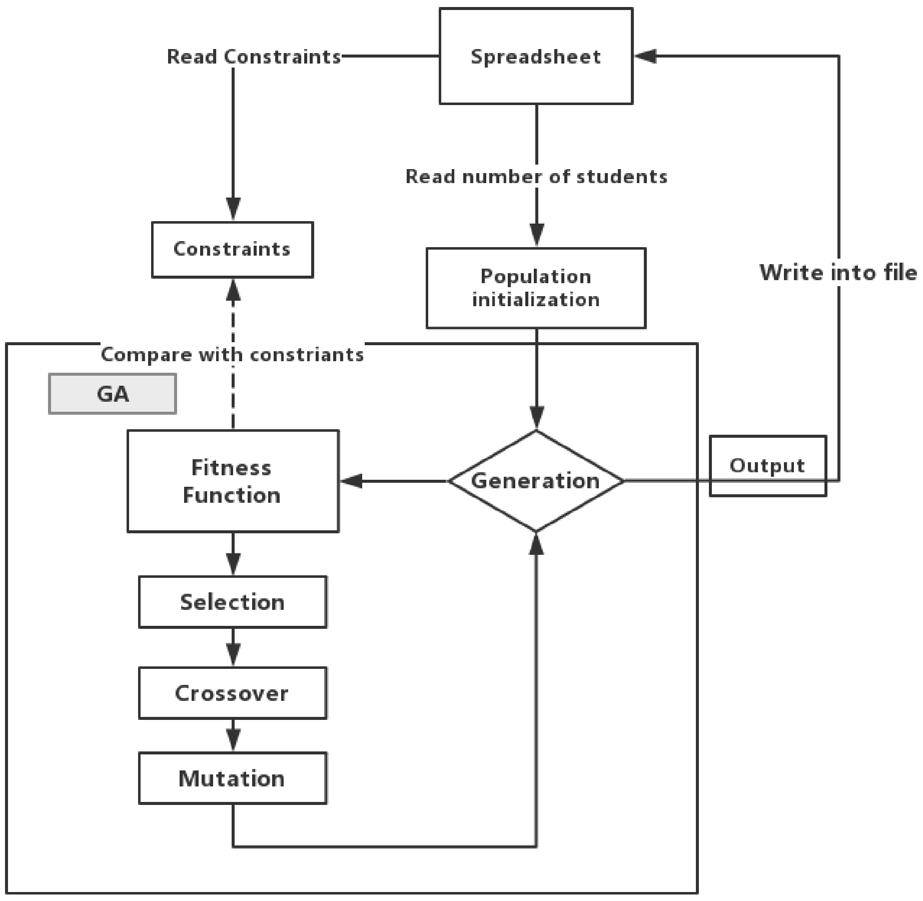


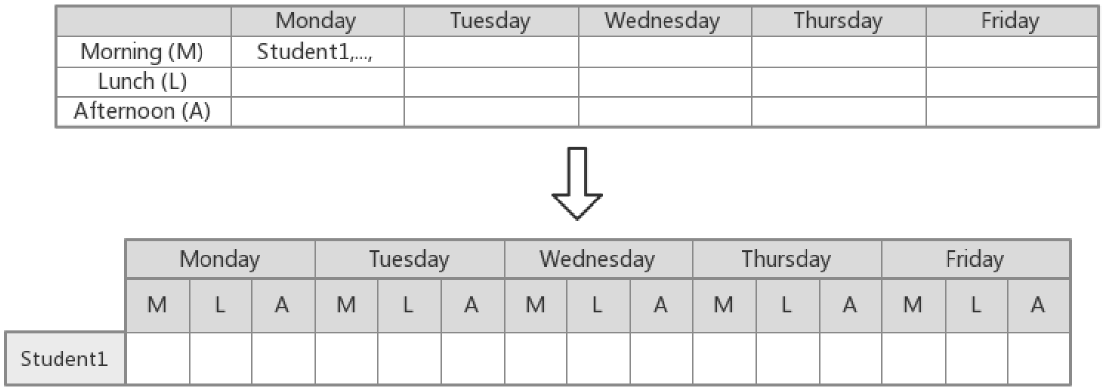
Figure 4.1: System Flow Diagram

4.3 Algorithmic Design

This section will design the algorithm according to the analysis of the requirements in Chapter 3. Specifically, the design of chromosome representation, fitness function, and special representation of constraints for computational fitness can be designed, and the method of selection, crossover and mutation in genetic algorithm is designed.

4.3.1 Chromosome Representation

A chromosome is a form of data representation of a target result when a genetic algorithm solves an optimization problem. As mentioned in section 3.1, the result of the goal is a complete timetable. However, such a timetable does not clearly indicate the constraints of the entire problem. Therefore, Figure 3.1 has been changed in the process of expressing chromosomes, as shown in 4.2. The horizontal axis of the original table refers to the day of weekdays, and the vertical axis identifies different sessions in each day, so each square in the content represents a specific session. For each specific session, the school can arrange different students, for example put student 1 in Morning session on Monday. However, in the revised timetable, the horizontal axis is indicated as a specific session. Due to five weekdays a week and each day is divided into three sessions: morning, lunch, and afternoon, the horizontal axis is 15 sessions. The vertical axis is expressed as a specific student. So in the new schedule, each line is determined as a student's respective timetable.



	Monday	Tuesday	Wednesday	Thursday	Friday
Morning (M)	Student1,...,				
Lunch (L)					
Afternoon (A)					

	Monday			Tuesday			Wednesday			Thursday			Friday		
	M	L	A	M	L	A	M	L	A	M	L	A	M	L	A
Student1															

Figure 4.2: Changed Schedule

The result is encoded using a binary code according to the analysis of the requirements in section 3.2. Combined with the new schedule expression mentioned above, the space will be filled with 1 or 0, 1 means that the student is scheduled in this session, and 0 is the opposite. As shown in Figure 4.3, each student's timetable can be represented as a 0/1 strings of length 15. Then, if there are N students are registered in the pre-school, the chromosome representation is a matrix of $N \times 15$.

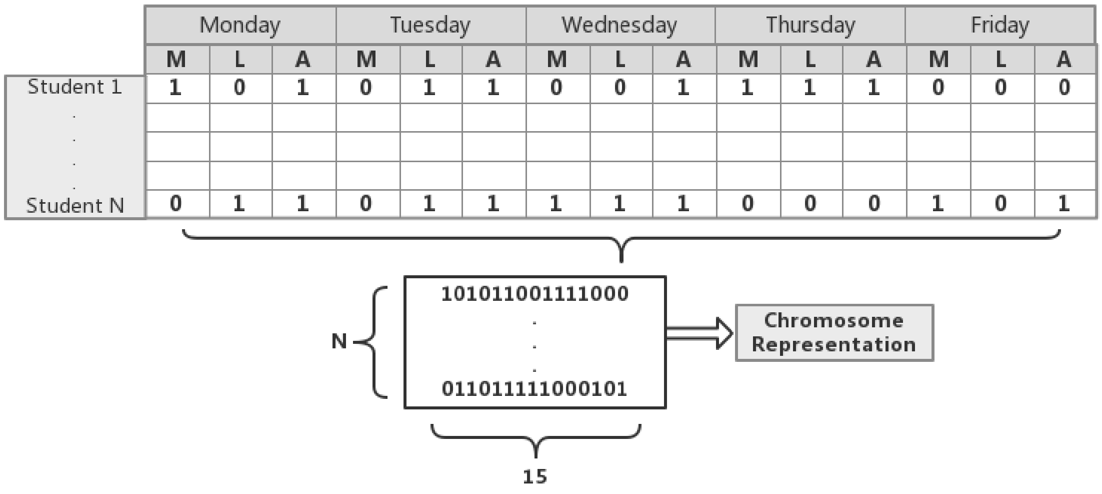
4.3.2 Constraints Design

There are a series of constraints in this project, which have been enumerated in Section 3.2. In order to embody them in the algorithm process, it is necessary to design the constraints in combination with the previously designed chromosome representation.

H1: Maximum number of students registered : 52.

In the algorithm calculation, N (the number of registered students) in the chromosome representation can be expressed as $N <= 52$.

H2: Maximum number of simultaneous students with session: 26.



	Monday			Tuesday			Wednesday			Thursday			Friday		
	M	L	A	M	L	A	M	L	A	M	L	A	M	L	A
Student 1	1	0	1	0	1	1	0	0	1	1	1	1	0	0	0
.															
.															
.															
Student N	0	1	1	0	1	1	1	1	1	0	0	0	1	0	1

Figure 4.3: Extract Chromosome Representation From New Schedule

In the chromosome representation (matrix of $N \times 15$), the number of simultaneous students for each specific session can be calculated by calculating the sum of each column. Then let this number be not larger than 26 as a condition of the algorithm.

H3: Can't choose lunchtime session alone.

Each line in the chromosome representation is a binary code of length 15. As shown in 4.4, if it is divided into 5 parts, each part is a binary code of length 3. For this binary code of length 3, there are 2^3 possibilities. Among such multiple possibilities, "010" may not be allowed to exist. The conversion of 010 to decimal is equal to 2. Therefore, for this constraint, it can be expressed that any part of the split can not be equal to 2.

H4: Student's weekly required attendance time (15 hours free + additional paid time).

This constraint can be first simply judged to accumulate the total scheduled time of each student's week, and in the chromosome representation is a horizontal summation. However, since the morning session and the afternoon session time are 2.5h, the noon session time is 1.25h. So it can't directly sum. It is necessary to perform weighting operations according to the duration of different sessions. For each student's string of binary codes, it can be divided into 5 parts as in H3. As shown in the 4.5, there are 8 possibilities for the binary code of length 3 in each part, and each case is calculated after 0h. The six results of 1.25h, 2.5h, 3.75h, 5h and 6.25h, so the five segments can

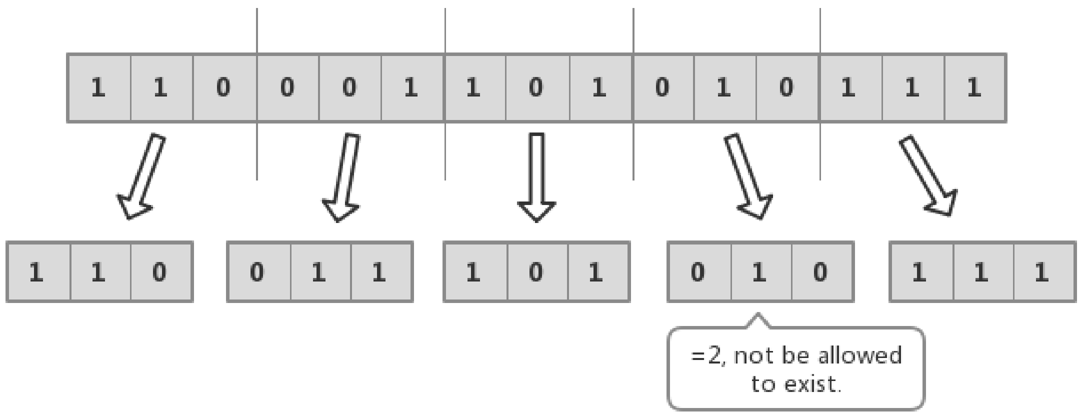


Figure 4.4: Single line chromosome representation segmentation (H3)

be calculated separately, and then the sum can be used to find the sum of the time each student is scheduled each week. This number is limited. Need to be equal to the length of time he requested.

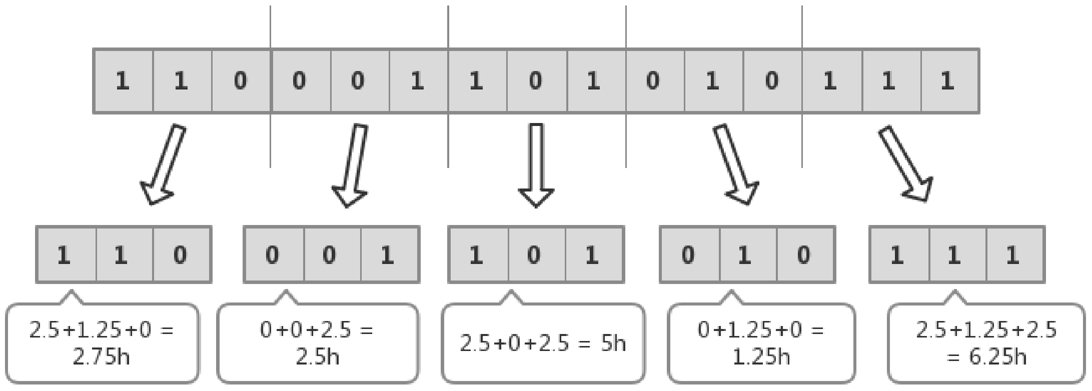


Figure 4.5: Single line chromosome representation segmentation (H4)

H5: Schedule only for newly enrolled children on each entry point. The original childs timetable remains unchanged if their parents do not have special requirements.

From the condition of this constraint, it can be analyzed that the entire genetic algorithm operation may be performed more than once in the entire algorithm calculation process.

Since the students are registered in batches, the size of N in the resulting matrix of $N \times 15$ varies, and N is always the latest number of registered students for this semester. However, when considering H1 and H2, as shown in 4.6, the size of N is the number of registered people in the previous semester, so when generating a schedule for newly registered M students, the vertical constraint H1 should satisfy the overall $N + M <= 52$, not just $M <= 52$. The calculation of the vertical constraint H2 should also be calculated from the overall calculation rather than just from the newly registered students.

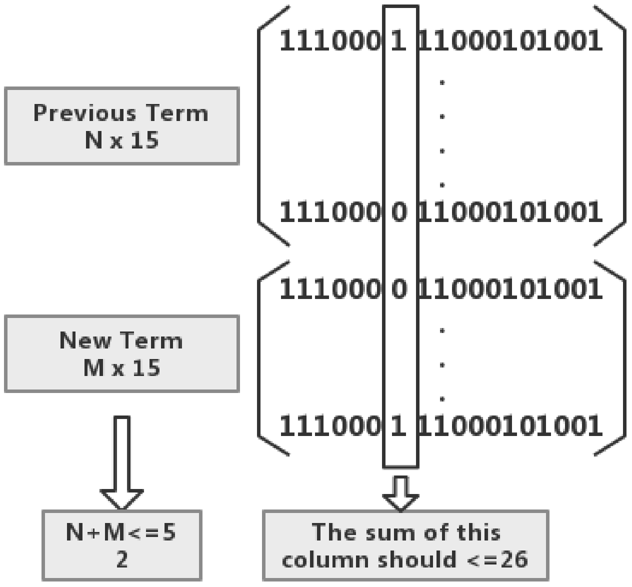


Figure 4.6: Overall constraint considerations for segmentation operations

S1: Parental Preferences

Parental preferences need to be reflected in the design of the algorithm. In the previous chapters, the types of parental preferences are introduced, including the day of weekdays, the morning or afternoon, or the morning or afternoon of the day of the weekdays. Even have multiple of these preferences. These preferences require a specific symbolic representation in the algorithm. So use a two-digit decimal number in the design to indicate a preference. The range of the first digit is 1-5, which means five weekdays. For example, '1' means Monday. The range of the second digit is 0 to 2, with '0' indicating no preference, '1' indicating morning, and '2' indicating afternoon. For example, '12' indicates that it tends to schedule care time on Monday afternoon. In the specific algorithm, each student's preference is regarded as a set P (preference), such as $P = \{11, 20, 32\}$, and then a set of facts R (real) is extracted according to the scheduled timetable for each possible result, If $P \subset R$, the preference is satisfied, and the opposite is not.

S2: Older students have higher priority

When the student is registered, the date of birth will be recorded as an attribute. In realizing this constraint, it is necessary to generate a schedule according to the order of birth date from early to late, that is, the order of age. The soft needs of older students need to be prioritized.

S3: Students with disabilities have the highest priority

When a student registers, students with disabilities will be marked with the highest priority, but the disability is not graded. Therefore, among students who are also disabled, priority will be determined according to age. According to such a setting, this constraint can be designed as a digital binary number, 0 means no disability, and 1 means disability. As it shows in 4.7, according to the previous constraints, the registered students of each semester are the divisional calculation schedule. According to this constraint, the registered students in each new semester can be divided into two parts, disabled and non-disabled. In general, the priority of disability is high, and in the two parts, they are arranged in order of age.

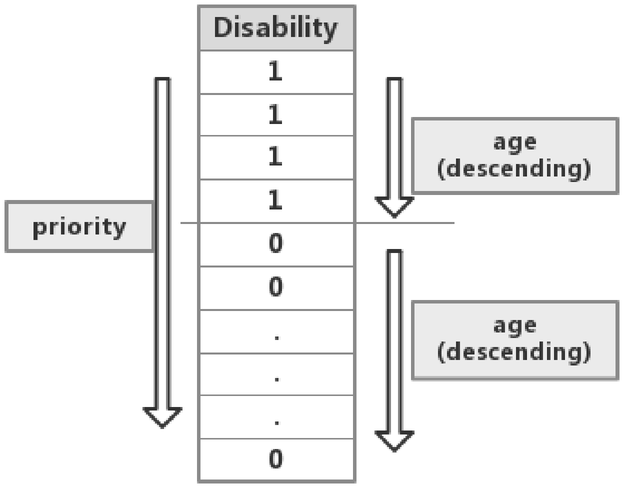


Figure 4.7: Priority Sequence Diagram

4.3.3 Fitness function

The fitness function is to calculate the fitness score of the result. The degree of fitness score reflects the degree of the result. So the design of the fitness function is very critical.In

the design process of fitness function, all constraints must be fully considered, but in order to make the process of system design more rapid and reasonable, some constraints will be added to the fitness function first, and then more constraints will be incrementally added after completing the whole framework. After initializing the population, each individual is a matrix of $N \times 15$. its fitness score will also be set to zero.
Due to H1 is an overall constraint, all should have been judged before the fitness function is calculated, so it will not be reflected in the fitness function. When considering H2, simply, each column is iterated for each matrix (individual) and summed for each column. When the number is not greater than 26, the fitness of the individual is increased by a value, and if the number is greater than 26, a value is decreased. When considering H3 and H4, iterate each line for each matrix, and divide and judge each line. If the duration meets the requirements of this student, this will increase the fitness score, otherwise it will decrease. If there is a fragment like '010', it will reduce the fitness score, otherwise increase. When H5 is added, the previous design for H2 needs to be modified. Firstly, calculate the sum of each column S_i in previous terms, then the sum of the i-th column in the new individual should be no larger than $26 - S_i$.
For S1, in the calculation of the fitness score, first analyze the session distribution of each row of the matrix, summarize the features of the distribution into a set R (real), and then read each student's preference set P. Then make a logical judgment, if $P \subset R$, increase the fitness score, otherwise reduce the fitness score. When considering S2 and S3, it can be considered to design a way to generate a schedule line by line according to the priority level. It is also possible to set a weight for each student according to the priority. When the preference of the student with high weight is satisfied, the higher fitness score is added, and when the preference of the low weight student is satisfied, the lower fitness score is increased.

4.3.4 Genetic Operators

Genetic operators include selection, crossover, and mutation, and their design will be described in detail below.

Selection Operator The role of the selection operator is to select better individuals in the population as a new generation of populations. An excellent individual is an individual with a high fitness score. In this project, the roulette selection method was chosen as the algorithm design for the selection operator. The roulette selection method, also called the proportional selection method, selects individuals according to the proportion of the individual in the whole. For example, when the population size is 10, the fitness score of each individual and the overall proportion are shown in Table 4.1. The pie chart is shown in Figure 4.8. It can be seen that the proportion of individuals with higher fitness scores in the circle is larger.In order to ensure the size of the population in the roulette selection method, the *Pop_size* iteration is performed according to the population size *Pop_size*, which is equivalent to throwing the ball to a disc similar to a pie chart. It is obvious that individuals with high fitness scores are

The chance of selection is relatively large. After selecting *Pop_size* times, the size of the population did not change, but the individuals in it have been updated, retaining individuals with higher fitness.

Individuals	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
Fitness Score	**23**	**102**	**10**	**250**	**400**	**267**	**328**	**91**	**409**	**200**

Table 4.1: Roulette Selection Method Sample Data

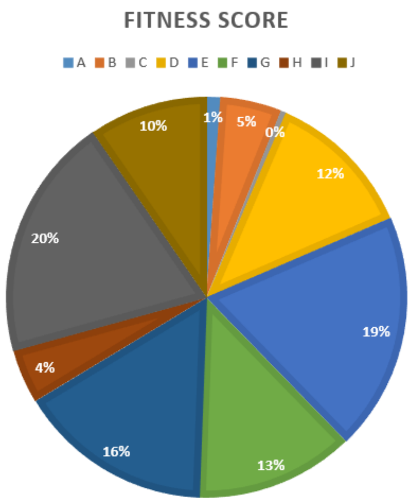


Figure 4.8: Proportional Distribution Pie Chart

Crossover Operator The crossover operator is an algorithm that mimics the recombination of DNA from two parents in the reproduction process. The common crossover algorithm has single point intersection and multi-point intersection. Single point cross can be selected as an algorithm design in this project. However, since the chromosome is not a common one-dimensional vector but a two-dimensional matrix, it is necessary to reset the rules when performing the crossover operation. Because each row of each individual's matrix is a student's weekly schedule, it can be regarded as an inseparable individual, so we can randomly select a row as a cross point, as shown in the figure 4.9, and then select two individuals as parents from the group. Then divide the two parents individuals from the intersection and recombine them to get the new two individuals, thus completing a crossover operation. Before implementing the crossover operation, the crossover rate needs to be set so that not all individuals in the population will crossover. The purpose of the crossed operators is to allow individuals in the population to exchange genes, expecting to produce better individuals.

Mutation Operator The role of the mutation operator is to cause mutations in individuals in the population. This mutation may be benign or malignant. A benign mutation may increase the individual's fitness score and be left in the process of

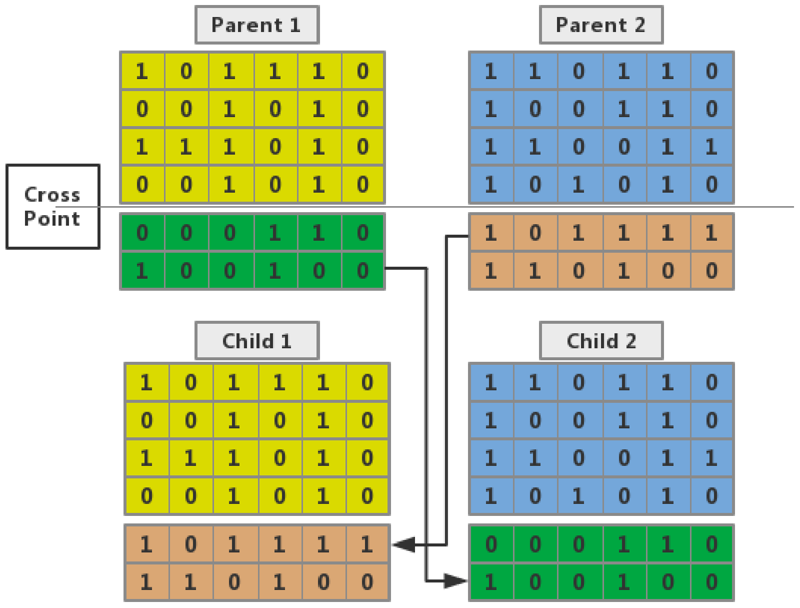


Figure 4.9: Single point cross schematic

selection. For binary code-encoded chromosome representations, a common method is to change a certain gene (a certain digit) of a selected individual from 0 to 1 or from 1 to 0. But as a two-dimensional matrix, the change will not be a digit, but a column number. As shown in the figure 4.10, a column of the selected individual is swapped between 0 and 1. At the same time, because the variation is a minority, it is also necessary to set a mutation rate to ensure that only a small number of individuals will mutate, and to retain as many of the elite individuals as possible while expecting benign mutations.

4.3.5 Termination conditions

In general, genetic algorithms need to set termination conditions to stop the iteration of the algorithm. The first approach is to set the boundary value by setting the fitness score of the optimal solution and using this score as the boundary value. An individual who has reached an fitness score to reach this boundary value during the iteration process can stop the iteration and output the individual as a result. Another way is to set the number of iterations, which is number of generations, and only stop at the end of this iteration. The individual with the highest fitness score is output as the result. Or combine these two methods, if there is an individual who reaches the boundary value during the iteration process, it will output directly, otherwise it will run until the end of the iteration and then output. Since the problem with this project is the NP-hard problem, it is difficult to find the optimal result.

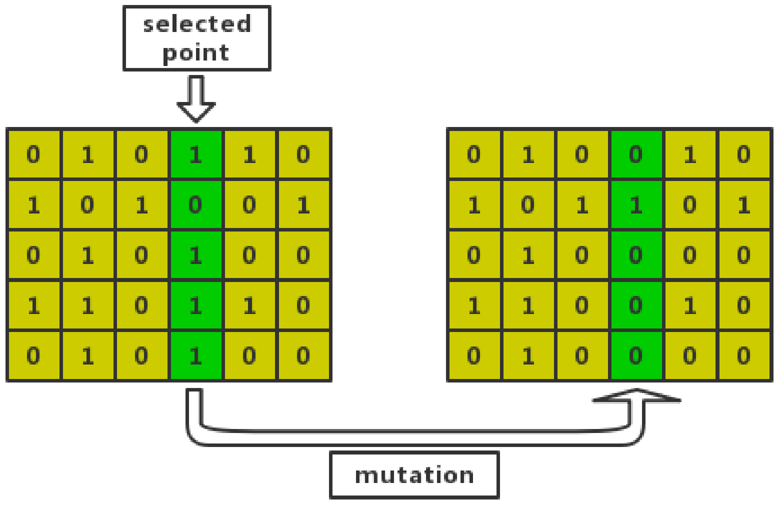


Figure 4.10: Mutation Schematic

So the second method will be used as a termination condition.

4.4 Summary

This chapter first determines that the design model of the entire project is a spiral development model, then each part of the system is designed according to the structure and each part is explained in detail. However, in the process of implementation, it is possible to adjust the design according to the actual situation, which is consistent with the idea of the spiral development model. The specific implementation will be described in the next chapter.

Chapter 5

Implementation and Testing

This chapter will describe the implementation of the system. Although the spiral development model is used in the actual implementation process, some functions are originally added to the system in an incremental manner, but this chapter will generally describe the functions implemented, and the specific structure will be similar to the division of the module design in Chapter 4. After that, the system will be tested and verified.

5.1 Implementation

Python is a high-level computer language with easy-to-read and concise code. At the same time support *Pandas* scientific computing library, *Numpy* matrix computing library and *openpyxl* library which can read data from Excel spreadsheet. Because of the simple syntax and numerous computational libraries, choosing a Python implementation can reduce the amount of programming time.

5.1.1 System input

As described in the design, the input and output of this system is the same excel spreadsheet file 5.1. The system input is to read each constraint and some parameters from this spreadsheet file, and processed into a data format that is easy to calculate. The main attributes of the spreadsheet entered as a system are shown in the table 5.1. The first column indicates the ID of the registered student, the second column indicates whether the student has a disability, 1 means yes, 0 or empty means no. The third column shows the student's schedule preferences, the specific meanings have been detailed in the design of the previous chapter. It is worth noting that the empty representative has no preference for the schedule. The fourth column indicates the student's date of birth and the fifth column indicates the student's registration time (semester). The subsequent color-coded content should have 15 columns, this is the timetable for each student. This setting is exactly the same as the design of the chromosome representation in the previous chapter. But for the sake of simplicity, it is reduced to three columns. The contents of these three columns are only 1 or 0 (empty), indicating whether

the student is scheduled to be In this particular session. The next column actually calculates the length of time each student in the line is actually scheduled each week. The last column records the weekly scheduled length of each student's request for registration.
There are three parts in color. Green represents the timetable that has been completed, which is the timetable for the previous term. Red identifies a timetable that has not yet been scheduled, but red is divided into three parts according to the Term column: autumn, spring and summer. Three parts should be filled in order instead of being calculated together. The yellow part refers to the summation of each column, because each column represents a specific session, and each session has an upper limit on the number of students, so the yellow part allows the user to have an intuitive understanding of the completion of the constraint.
After understanding the format of the input, the data that needs to be read from the table is as follows:

1. Read the data of students in the autumn (or spring, or summer) in Pandas' *DataFrame* format. Take the fall as an example. As shown in the table 5.2, the gray part of the data (empty will be filled with 0) should be read and stored.

2. The yellow grid will be read and stored as an array format and will be used as a vertical constraint.

					am	lunch	pm		
ID	Disability	Preference	Date of Birth	Term	3	1	4	Hours arranged	Hours Required
1	1	21	2013/9/12	PREV	1		1		12.5
2	0	30	2013/10/15	PREV	1		1		15
3	0		2013/12/13	PREV		1	1		12.5
4	1	52	2013/11/22	PREV	1		1		15
5	0		2014/2/9	Autumn					15
6	0	30	2014/1/6	Autumn					15
7	0		2014/5/29	Spring					17.5
8	0	11	2014/6/11	Spring					15
9	1		2014/6/2	Summer					15
10	1		2014/5/9	Summer					15

Table 5.1: Input spreadsheet example

Term
Autumn
Autumn

ID	Disability	Preference	Date of Birth	Hours Required
5			2014/2/9	15
6	0	30	2014/1/6	15

Table 5.2: Autumn input sample

5.1.2 Parameter setting

Some parameters need to be set during algorithm implementation. As shown in the table 5.3, These settings are based on default values set by previous experience. and will be optimized after the entire system is completed.

Parameters	**Set value**
Population Size(P)	**100**
Number of Generation(G)	**200**
Cross Rate	**0.4**
Mutation Rate	**0.02**

Table 5.3: Parameters setting

5.1.3 Initialization of Population

In the design of the chromosome representation in Chapter 4, it has been clarified that the representation of a chromosome is a matrix of size $N \times 15$ with only 0 or 1. The N should be set based on the number of new semester enrollments in the read input. A population is a collection of individuals with a population size (P). As shown in the figure 5.1, a three-dimensional array will be used in the process of code implementation. The specific code is as follows.

```
self.pop = np.random.randint(2, size=(pop_size, N, chromlen))
```

Randomly fill 0 or 1 in the 3D matrix of $P \times N \times 15$.

5.1.4 Constraint implementation

Through the analysis of the problem, it was found that it is easier to implement hard constraints and some soft constraints, the detail will be discussed in the fitness function in the next section. but it is difficult to implement the age-based priority. Because the genetic algorithm is a random-based optimization algorithm, and it is also based on the individual, in this problem the individual is a complete timetable rather than a student's timetable (because there are constraints in the vertical, it cannot be split) . Therefore, in the fitness function, the fitness score can be set to 0 to allow some unwanted individuals to be eliminated, or to set a higher fitness score to allow good individuals to be left behind. However, it is difficult to achieve priority setting within an individual. In simple terms, genetic algorithms can roughly achieve the existence or nonexistence of certain features, but not in a certain order.
In order to solve the problem of priority as much as possible, I decided to use weights to add a weighted attribute to each student, as shown in the table 5.4. This attribute is a number whose size is first based on the student's age, and then if there is a disability attribute, the student will increase the weight. Code show as below, First generate an evenly enlarged array

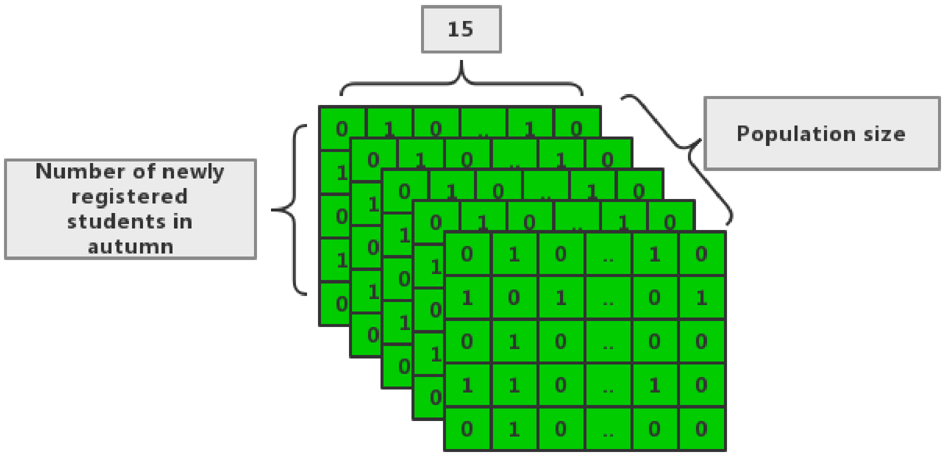


Figure 5.1: Population initialization diagram

of size D, then multiply the number of the 'Disability' column by 10 and add it to D. Due to the number of 'Disability' is only 0 or 1, only if students have disabilities, their weight will increase. The gray data in the table 5.2 is then sorted in order of birth date from morning to night, and D is added to the already sorted matrix as a new column. Finally, reorder them in the order of 'ID' to restore the original order.

```
D = np.arange(0.0, 10.0, 10 / N).round(2)[::-1]
x = d3['Disability'] * 10
Data['Weight'] = d5 + x
```

ID	Disability	Preference	Date of Birth	Hours Required	Weight
5	0		2014/2/9	15	7
6	0	30	2014/1/6	15	3

Table 5.4: Add weight attribute

5.1.5 Implementation of fitness function

The function of the fitness function is to calculate the fitness score for each individual of the population, so the input of the fitness function should be the entire population, and the output should be a one-dimensional array of length P, and each number in the array is the generation of the population. The fitness score for each individual. According to the design of the fitness function, first initialize an array of all 0s of length P, and then start iterating each individual in the population. In each iteration, the current individual's fitness score will be increased or decreased according to the horizontal and vertical constraints. The specific pseudo code is as follows:

```
def fitness(self):
    result = np.zeros(self.pop_size) # initialization
    #Iterating the individuals in the population
    for individual in range(self.pop_size):
        #Iterate through each row of the individual
        #(considering horizontal constraints)
        for line in individual:
        totalhours = 0
        perference = []
         #Divide each line into five segments.
         #Each segment is a string similar to '101'.
        timeslots = np.split(line, 5)
        for num in range(5):
        #Calculate the total duration and
        #calculate the course features of each day
            totalhours += duration of each day
            perference.append(Course features of each day)
        if Duration meets requirements or
        perference contains preferences of constraints
                    result[individual] increase
        else:
            result[individual] decrease
        for column in table.T:
            if sum of column + sum of previous column > 26:
                #Set to 0 to remove this individual
                result[individual] = 0
    return result
```

5.1.6 Implementation of genetic operators

All the operators and fitness functions are placed in one class.The whole process of the genetic algorithm can be implemented by creating a new entity of this class in the main function. The implementation of each operator is as follows:

Selection: The selection operator is completely implemented according to the design of the previous chapter, so the roulette selection method is used. Python's random library has a similar method based on weight selection, so the code is as follows:

```
def select(self):
    fitness = self.fitness() + 0.0000001
    a = np.arange(self.pop_size)
    rate = fitness/fitness.sum()
    i=random.choice(a,size=self.pop_size,replace=True, p=rate)
    return self.pop[i]
```

First calculate the fitness score of the whole population. In particular, if the fitness score is 0, an error will be reported, so add a small number to each number to ensure that each individual's fitness score is greater than 0. Then from P individuals select P individuals (each individual's fitness score as a weight) .

Crossover: In the previous chapter, the single-point intersection used for the crossover operator design, but in order to achieve better results in the implementation process, the multi-point intersection method was applied. As shown in the figure 5.2, Multi-point crossover is to randomly select half of the rows in the parent 1's chromosome, and replace these rows corresponding to the parent 2 to generate the child.In the implementation process, the crossover rate also needs to be considered, so the traversal is first performed for each individual, and then a random number is generated. If the random number is smaller than the crossover rate, the above-mentioned crossover process is started. Pseudo code as below.

```
def crossover(self, parent, pop):
    if random.rand() < cross_rate:
      individual = random.randint(0, pop_size, size=1)
      cross_points=random.randint(0, N, N/2)
      parent[cross_points]=pop[random_individual, cross_points]
    return parent
```

Mutation: The implementation of the mutation is also in accordance with the design of the previous chapter. However, since H3 mentions that students can't just choose a lunch session, when performing column mutations, only choose columns from non-lunch session columns for mutation. In addition, when performing the exchange function of 0 and 1, I did not directly use the exchange method, but reduced this column by 1, and then the absolute value. Such an operation can make 1 become 0 and let 0 become -1. Then it becomes 1. Although it is not an exchange behavior, it has the same result as the exchange. The pseudo code is as follows.

```
    def mutate(self, child):
        for point in replace_list:
            if random.rand() < mutate_rate:
                child.T[point] = abs(child.T[point] - 1)
        return child
```

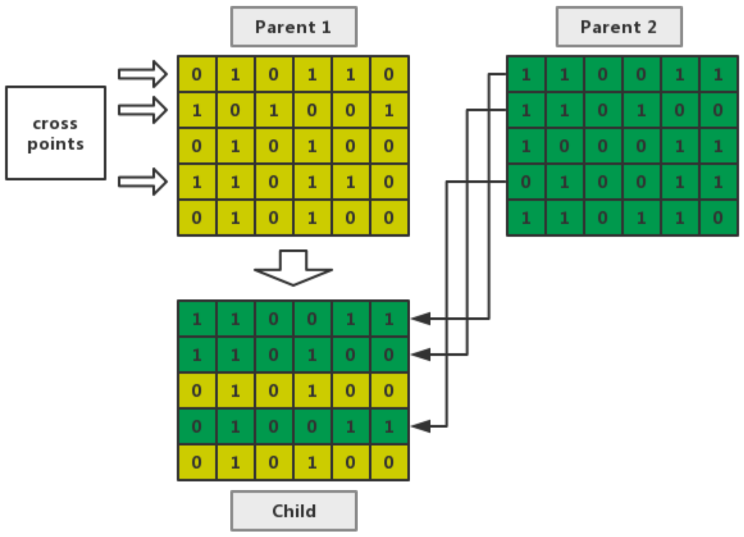


Figure 5.2: Multi-point cross schematic

evolution: Evolution is not a genetic algorithm is an operator, but a combination of individual operators for a complete genetic evolution process. The pseudo code is as follows.

```
def evolve(self):
    pop = self.select()
    pop_copy = pop.copy()
    for parent in pop:
        child = self.crossover(parent, pop_copy)
        child = self.mutate(child)
        parent[:] = child
    self.pop = pop
```

5.1.7 System output

After the last iteration of the system, the individuals with the highest fitness scores in the population will be selected. Since each individual is a two-dimensional array in the form of a timetable, all of them can eventually be written directly into the spreadsheet. This allows the achievement of various constraints in the horizontal and vertical directions to be visualized.

5.2 Testing

This section will use unit testing and integration testing to test the functionality of the system.

5.2.1 Unit Test

Unit testing is to test the function of each small module in the system. In this project, according to the design of the module division, it will test functions such as crossover, mutation, calculation of fitness scores, etc. The specific results are shown in the table 5.5.

Functions	testable	Pass
Input	$\surd$	$\surd$
Fitness Function	$\surd$	$\surd$
Crossover	$\surd$	$\surd$
Mutation	$\surd$	$\surd$
Output	$\surd$	$\surd$

Table 5.5: Unit Test Result

In the test process, due to the randomness of the genetic algorithm, it is necessary to replace part of the random number with a certain number to ensure that the result is predictable. Detailed test code can be found in the **Appendix**.

5.2.2 Integration Test

Integration testing in this project is to ensure that the entire system can run successfully and produce results. That is, the modules of the unit test are combined for functional testing. Since the input and input modules have been tested separately in unit tests, the implementation of the entire genetic algorithm will be primarily tested here. The figure 5.3 below shows the generation curve of the fitness score for the results using a genetic algorithm. The figure shows that the genetic algorithm is stable to a relatively high degree of fitness after a certain number of iterations. This shows the feasibility of this system. Since the entire schedule is segmented, that is, divided into three new terms, the three lines in the figure represent timetables for three different terms.

5.3 Summary

This chapter completely describes the implementation of each module of the system, and gives the pseudo code of the key parts. At the same time, the parts and the whole were tested. The analysis of the results and some of the problems encountered during the implementation will be discussed in the next chapter.

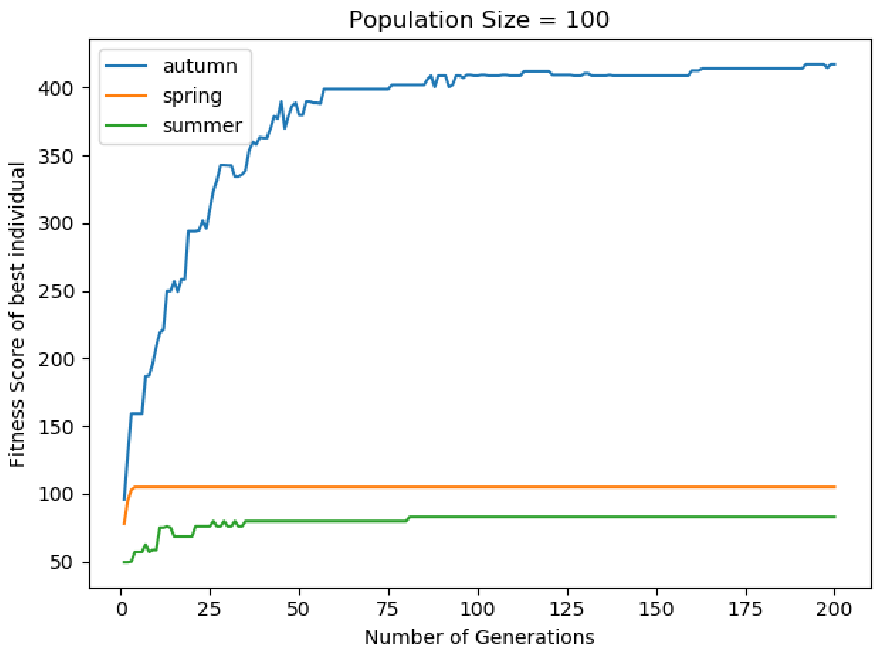


Figure 5.3: Integration test results

Chapter 6

Evaluation and Discussions

This chapter will evaluate and implement the system and critically analyze the process of algorithm selection and implementation. First, the parameters need to be optimized before the evaluation. The parameters mentioned in this project are mentioned above. As shown in the table 5.3, these parameters will affect the efficiency and result of the algorithm. In order to get better results and reduce running time.

6.1 Parameters Optimization

The control variable method will be used when optimizing the parameters, that is, controlling other variables to be constant, and finding a suitable value by changing the single variable. However, there are several parameters in this problem. Because of the possible mutual influence between the parameters, in order to minimize the influence of the parameters optimized first on the parameters optimized later. Therefore, the parameters should be optimized in order according to the degree of influence on the results, that is, the parameters with larger influences are optimized first.

Generations : The number of generations is the parameter that has the greatest impact on the running time of the entire algorithm, but if there is not enough large number of generations, it is impossible to optimize a sufficiently good individual, which means that the result cannot meet the requirements. So in order to weigh the optimal conclusion between the running time and the outcome, the following test process will be practiced. The following two figures 6.1 and 6.2 show the change curves of the number of generations from 0 to 200 at the intersection rate and the mutation rate, and the population size is 500 and 1000 respectively. As mentioned in the previous chapter, because the input is three different semester, there are three different curves of the colors are represented separately.
It can be seen from these two figures that the fitness score increases rapidly in the process of increasing the number of generations from 0 to 50, and gradually becomes stable when it reaches about 100. So for the number of generations, the choice of 100 is a reasonable setting.

Population Size :The size of the population will also be a key parameter affecting the

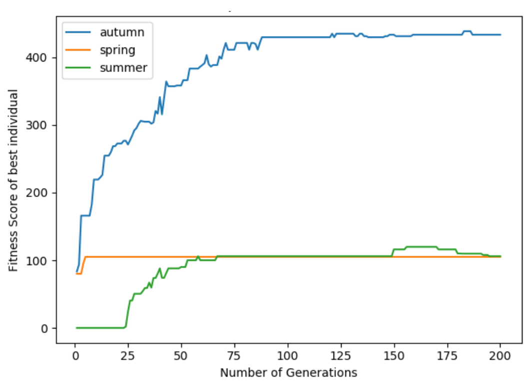


Figure 6.1: Fitness score with generations (Population Size = 500)

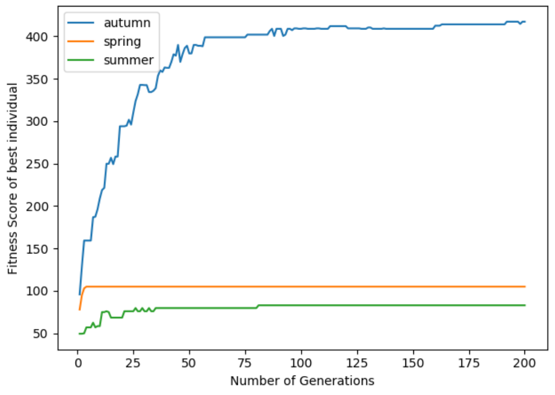


Figure 6.2: Fitness score with generations (Population Size = 1000)

results. As shown in the previous two figures, when the population size is different, the fitness score of the final result is different. So the figure 6.3 shows the plot of the population size from 100 to 1000 when the number of generations is set to 100.
It can be seen from the figure 6.3 that the fitness score of the results will be lower when the population size is below 200, and will stabilize after 200, but there will be some fluctuations, and still maintain a small increase after 600. This suggests that the larger the population, the better the optimization of the results. However, considering the test time, 500 will be selected as the optimized parameter setting for the population size.

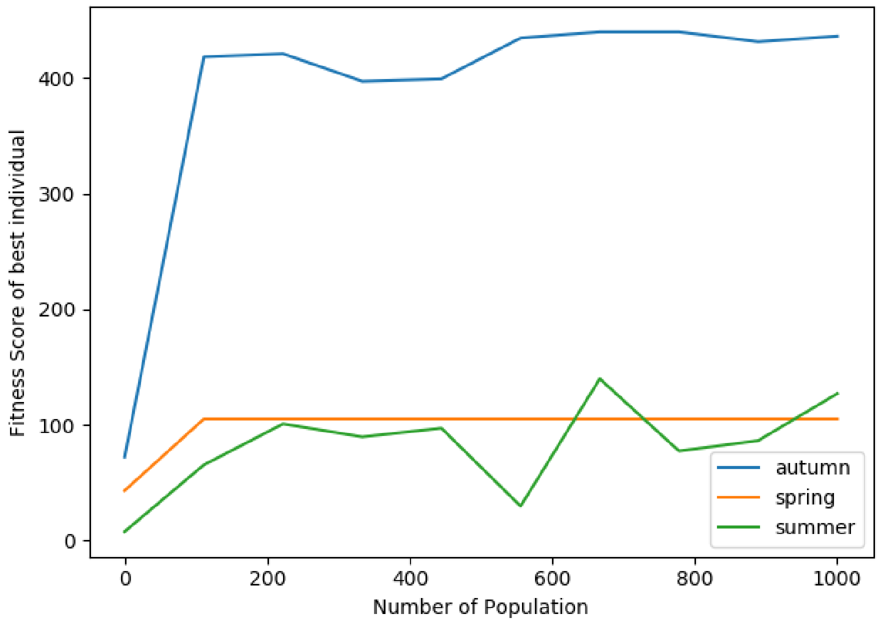


Figure 6.3: Curve of fitness score with population size (generation = 100)

Crossover Rate and Mutation Rate :The crossover rate and the mutation rate are internal parameters of the genetic algorithm. Although there are also effects on the algorithm results, since the initial set values are relatively conventional settings [8], these two parameters are optimized after the population size and the number of generations. In these two parameters, the influence of the crossover rate is significantly greater than the variability, so the size of the crossover rate is prioritized. As shown in the figure 6.4 , the number of generations and population size of the two tests were 100 and 500 that were previously optimized. However, since the whole algorithm is segmented for the generation of the schedule, this means that the subsequent term will become more and more demanding due to the increase in the number of newly registered students in the previous terms, and it is more and more difficult to achieve. So compared to the autumn term, the spring term and summer term curves are less obvious. It can be seen in this figure that the higher the crossover rate, the better the results will be better. However, considering the randomness factor, the crossover rate can be set to 0.8. This not only ensures a high crossover rate to optimize the results, but also avoids the 100% crossover operation to allow the outstanding individuals in the population to be changed.

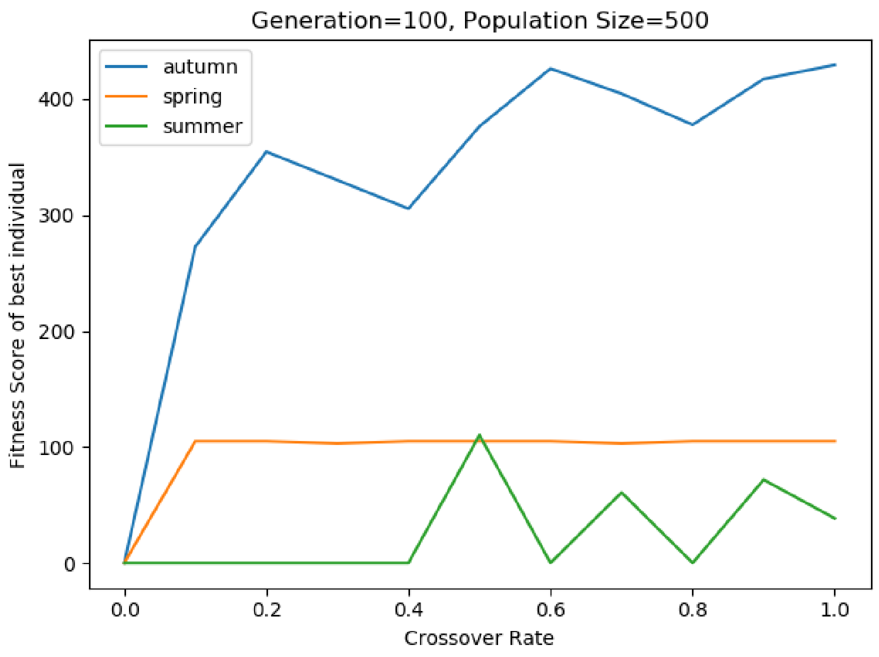


Figure 6.4: Curve 1 of fitness score with crossover rate (population size = 500, generation = 100)

The mutation rate is optimized after the crossover rate is determined. The general mutation rate setting will be lower because the probability is actually lower according to the natural law. However, in order to set the appropriate mutation rate according to the actual problem, the range of the mutation rate is first set to $[0, 1]$ in the test process, but the result is a bowl-shaped curve, which means that the mutation rate is not monotonously increased.

Therefore, according to common sense, the range of the mutation rate is set to $[0, 0.1]$, and the test results are shown in the figure 6.5. It can be seen from the figure that the fitness score of the results is relatively high when the mutation rate is between 0.05 and 0.06. Therefore, 0.06 will be set to the optimized mutation rate.

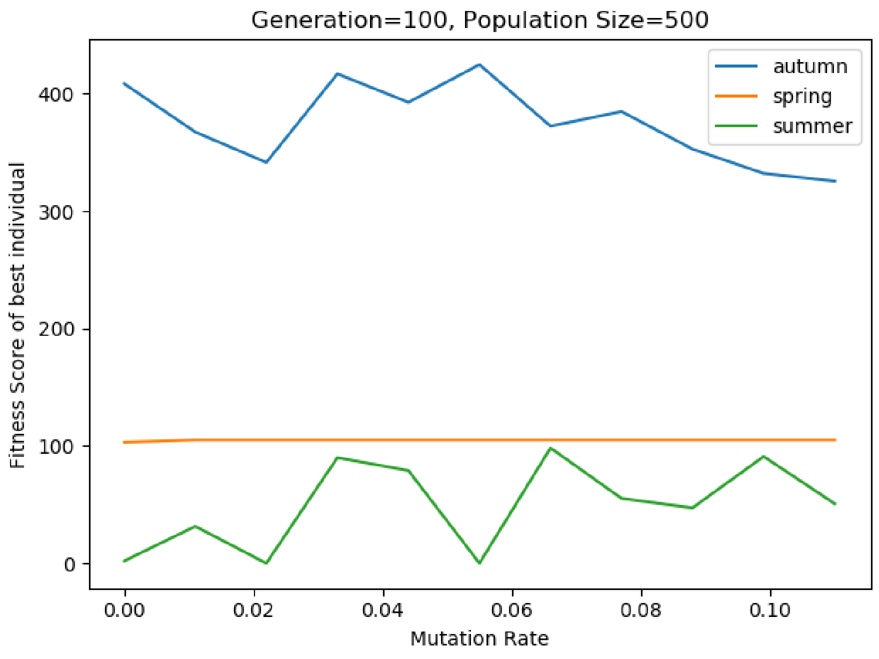
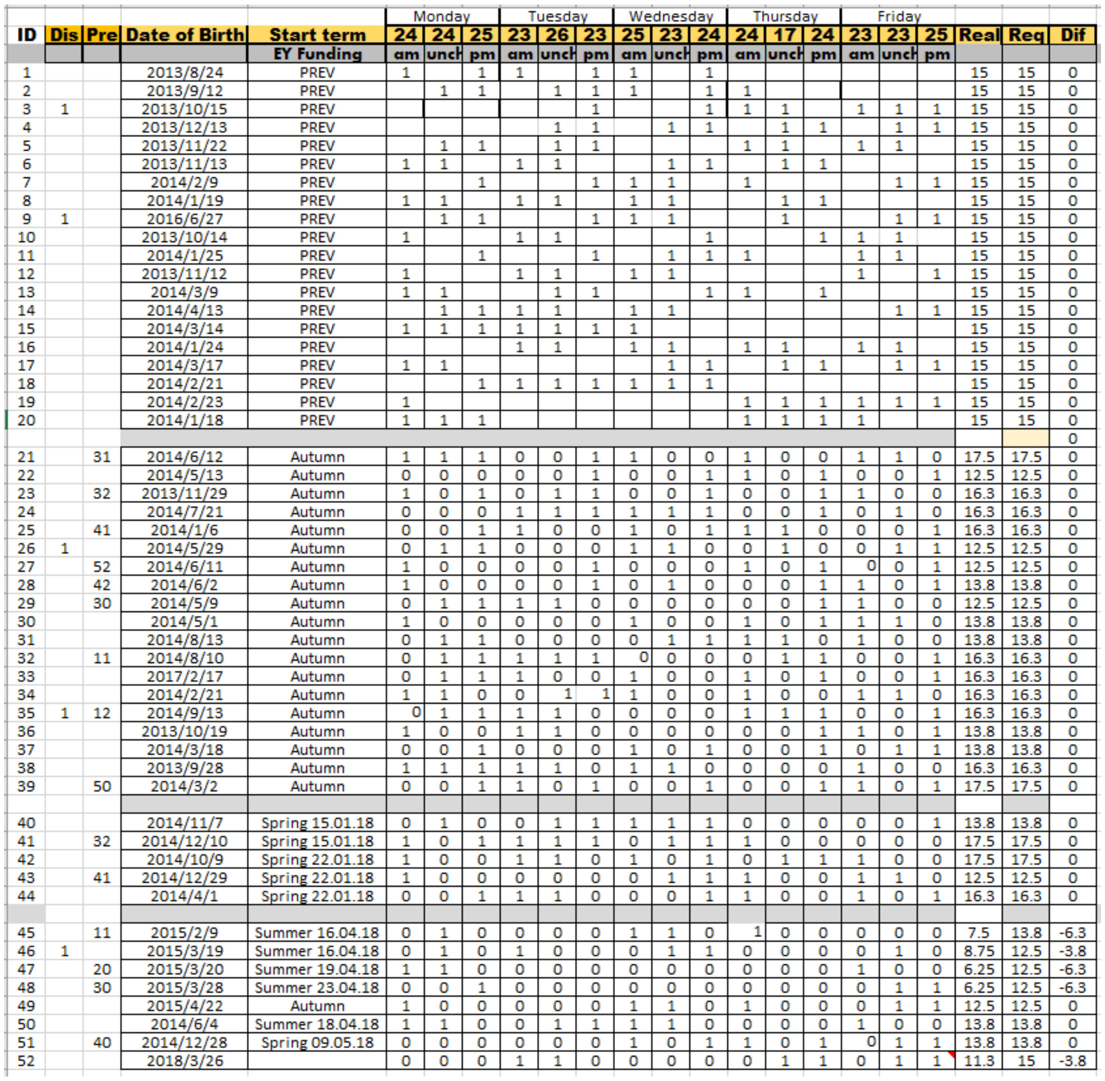


Figure 6.5: Curve of fitness score with mutation rate (population size = 500, generation = 100)

6.2 Result

After optimizing each parameter, the operation is performed as shown in the figure 6.6. From the second line in the figure, it is can find that the vertical constraint H2 of the result is all achieved, because all the numbers are not greater than 26. In addition, from the last column (the difference between the actual time and the required time), it can be seen that the time requirements of the autumn term and spring term students have all been reached, because they are all 0. However, the summer term is not fully satisfied. As mentioned earlier, this is because the total number of registered students has reached 52, so for the last few registered students, the constraints for scheduling them are very demanding, which may lead to the inability to find suitable the result of. But overall, this system has successfully solved the scheduling problem.



					Monday			Tuesday			Wednesday			Thursday			Friday					
ID	Dis	Pre	Date of Birth	Start term	24	24	25	23	26	23	25	23	24	24	17	24	23	23	25	Real	Req	Dif
				EY Funding	am	unch	pm	am	unch	pm	am	unch	pm	am	unch	pm	am	unch	pm			
1			2013/8/24	PREV	1		1	1		1	1		1							15	15	0
2			2013/9/12	PREV		1	1		1	1	1		1	1						15	15	0
3	1		2013/10/15	PREV						1			1	1	1		1	1	1	15	15	0
4			2013/12/13	PREV					1	1		1	1		1	1		1	1	15	15	0
5			2013/11/22	PREV		1	1		1	1				1	1		1	1		15	15	0
6			2013/11/13	PREV	1	1		1	1			1	1		1	1				15	15	0
7			2014/2/9	PREV			1			1	1	1		1				1	1	15	15	0
8			2014/1/19	PREV	1	1		1	1		1	1			1	1				15	15	0
9	1		2016/6/27	PREV		1	1			1	1	1			1			1	1	15	15	0
10			2013/10/14	PREV	1			1	1				1			1	1	1		15	15	0
11			2014/1/25	PREV			1			1		1	1	1			1	1		15	15	0
12			2013/11/12	PREV	1			1	1		1	1					1		1	15	15	0
13			2014/3/9	PREV	1	1			1	1			1	1		1				15	15	0
14			2014/4/13	PREV		1	1	1	1		1	1						1	1	15	15	0
15			2014/3/14	PREV	1	1	1	1	1	1	1									15	15	0
16			2014/1/24	PREV				1	1		1	1		1	1		1	1		15	15	0
17			2014/3/17	PREV	1	1						1	1		1	1		1	1	15	15	0
18			2014/2/21	PREV			1	1	1	1	1	1	1							15	15	0
19			2014/2/23	PREV	1									1	1	1	1	1	1	15	15	0
20			2014/1/18	PREV	1	1	1							1	1	1	1			15	15	0
																						0
21		31	2014/6/12	Autumn	1	1	1	0	0	1	1	0	0	1	0	0	1	1	0	17.5	17.5	0
22			2014/5/13	Autumn	0	0	0	0	0	1	0	0	1	1	0	1	0	0	1	12.5	12.5	0
23		32	2013/11/29	Autumn	1	0	1	0	1	1	0	0	1	0	0	1	1	0	0	16.3	16.3	0
24			2014/7/21	Autumn	0	0	0	1	1	1	1	1	1	0	0	1	0	1	0	16.3	16.3	0
25		41	2014/1/6	Autumn	0	0	1	1	0	0	1	0	1	1	1	0	0	0	1	16.3	16.3	0
26	1		2014/5/29	Autumn	0	1	1	0	0	0	1	1	0	0	1	0	0	1	1	12.5	12.5	0
27		52	2014/6/11	Autumn	1	0	0	0	0	1	0	0	0	1	0	1	0	0	1	12.5	12.5	0
28		42	2014/6/2	Autumn	1	0	0	0	0	1	0	1	0	0	0	1	1	0	1	13.8	13.8	0
29		30	2014/5/9	Autumn	0	1	1	1	1	0	0	0	0	0	0	1	1	0	0	12.5	12.5	0
30			2014/5/1	Autumn	1	0	0	0	0	0	1	0	0	1	0	1	1	1	0	13.8	13.8	0
31			2014/8/13	Autumn	0	1	1	0	0	0	0	1	1	1	1	0	1	0	0	13.8	13.8	0
32		11	2014/8/10	Autumn	0	1	1	1	1	1	0	0	0	0	1	1	0	0	1	16.3	16.3	0
33			2017/2/17	Autumn	0	1	1	1	0	0	1	0	0	1	0	1	0	0	1	16.3	16.3	0
34			2014/2/21	Autumn	1	1	0	0	1	1	1	0	0	1	0	0	1	1	0	16.3	16.3	0
35	1	12	2014/9/13	Autumn	0	1	1	1	1	0	0	0	0	1	1	1	0	0	1	16.3	16.3	0
36			2013/10/19	Autumn	1	0	0	1	1	0	0	0	0	0	0	1	1	0	1	13.8	13.8	0
37			2014/3/18	Autumn	0	0	1	0	0	0	1	0	1	0	0	1	0	1	1	13.8	13.8	0
38			2013/9/28	Autumn	1	1	1	1	1	0	1	1	0	0	0	0	1	0	0	16.3	16.3	0
39		50	2014/3/2	Autumn	0	0	1	1	0	1	0	0	1	0	0	1	1	0	1	17.5	17.5	0
40			2014/11/7	Spring 15.01.18	0	1	0	0	1	1	1	1	1	0	0	0	0	0	1	13.8	13.8	0
41		32	2014/12/10	Spring 15.01.18	1	0	1	1	1	1	0	1	1	1	0	0	0	0	0	17.5	17.5	0
42			2014/10/9	Spring 22.01.18	1	0	0	1	1	0	1	0	1	0	1	1	1	0	0	17.5	17.5	0
43		41	2014/12/29	Spring 22.01.18	1	0	0	0	0	0	0	1	1	1	0	0	1	1	0	12.5	12.5	0
44			2014/4/1	Spring 22.01.18	0	0	1	1	1	0	0	0	1	1	0	0	1	0	1	16.3	16.3	0
45		11	2015/2/9	Summer 16.04.18	0	1	0	0	0	0	1	1	0	1	0	0	0	0	0	7.5	13.8	-6.3
46	1		2015/3/19	Summer 16.04.18	0	1	0	1	0	0	0	1	1	0	0	0	0	1	0	8.75	12.5	-3.8
47		20	2015/3/20	Summer 19.04.18	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	6.25	12.5	-6.3
48		30	2015/3/28	Summer 23.04.18	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	6.25	12.5	-6.3
49			2015/4/22	Autumn	1	0	0	0	0	0	1	1	0	1	0	0	0	1	1	12.5	12.5	0
50			2014/6/4	Summer 18.04.18	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0	13.8	13.8	0
51		40	2014/12/28	Spring 09.05.18	0	0	0	0	0	0	1	0	1	1	0	1	0	1	1	13.8	13.8	0
52			2018/3/26		0	0	0	1	1	0	0	0	0	0	1	1	0	1	1	11.3	15	-3.8

Figure 6.6: The input file that written the result

6.3 Result Evaluation

So far, the entire system has been tested to indicate that it can run. Then the individual parameters have also been optimized. A more comprehensive evaluation of the system is now required. Robustness is the main direction of the evaluation system. The method may be by randomly generating duration requirements, number of students, preferences, disability, and giving different generation ratios, and then evaluating the performance of the system through the results.

Firstly, change the student's duration requirements, based on the 15-hour free time, randomly increase the duration or decrease the duration, and increase the duration and decrease the duration to meet the duration of the session. Then, it will be divided into ten parts in a ratio of 0 to 1, that is, 0%, 10%, ..., 90%, 100% . Then observe whether the system can meet the appropriate target for different proportions of random duration, that is, whether the fitness score can reach the expected level. The figure 6.7 shows the curve of the fitness score with the number of generations when the 60% of students' duration requirements are randomly generated. It can be found from the figure that the whole optimization process is similar to the previous integration test, which shows that even if the randomly generated constraint is used, it can obtain an expected result. Figure 6.8 shows the results of different scale generations. It can be seen from the figure that the fitness scores of the students in the autumn and spring are relatively stable within a relatively good range of results. However, the fitness scores of summer students' schedules vary widely, sometimes higher and sometimes lower. Through the structural analysis of the input data, the reason for this result is as follow. Autumn term and spring term, especially in the autumn, their timetable is prioritized, which is in line with the constraint H5, so the vertical constraints are relatively loose when the students are scheduled for the two semesters. However, as the number of students who have been arranged increases, the later semester will have more stringent constraints in the vertical direction, which makes the algorithm process not easy to find better results, making the problem of randomness magnified. Therefore, when scheduling students in the summer, it is prone to instability.

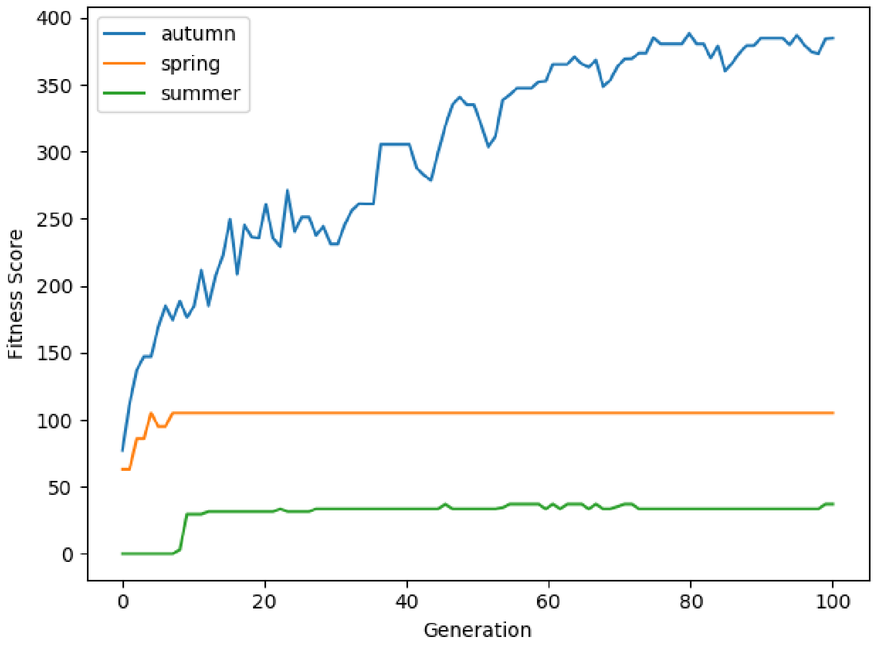


Figure 6.7: Fitness score curve (randomly generated 60% time requirement)

Secondly, randomly generate student preferences, randomly add preferences to students of different proportions, and then test the fitness scores of the generated results. As previously required for the duration, it is divided into random proportions of 0 to 100%, and the results

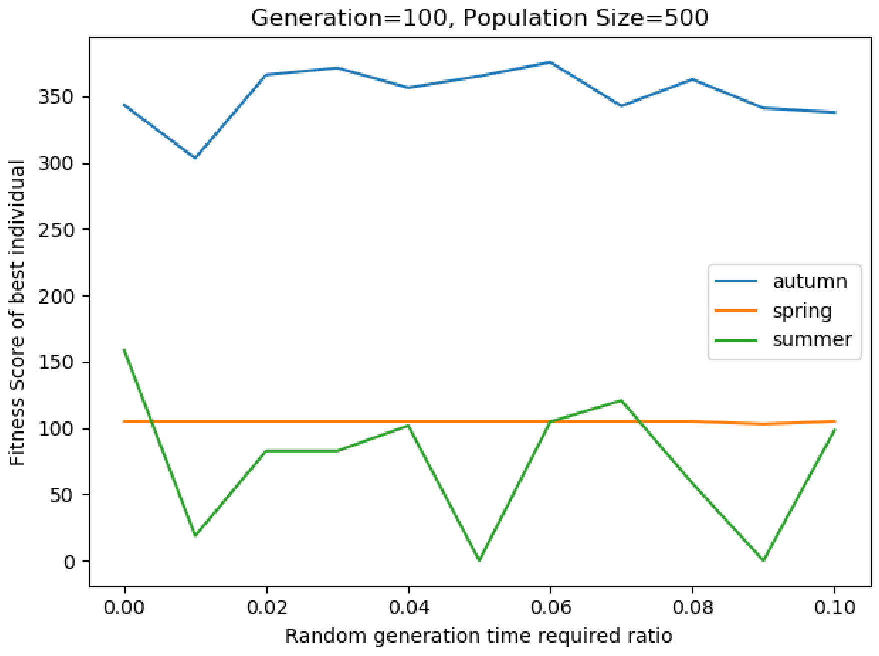


Figure 6.8: Fitness score curve (time requirement randomly generated from 0 to 100%)

are shown in the figure 6.9. As the fitness score increases slightly with the increase of the preference ratio, this is because as the student's preference increases, the process of finding the right result (to optimize the result) is more difficult. At the same time, the fitness of the summer timetable is still unstable, for reasons explained earlier. The system could have been actually tested by submitting it to the end user, but since it is currently the summer vacation of these institutions, the staff is not currently in the position, so the actual use is not possible. If time is sufficient, the product can be delivered and tested later. In this way, further corrections and improvements are made.

6.4 Discussion

Based on the above evaluation of the results, it can be determined that the system can completely solve the requirements of only basic constraints, or basically solve the requirements of complex constraints. But there are also some areas that can be improved.

First, for the initialization of the population, since the initialization is completely random in this system, which leads to the initial individual's fitness score may be very low. It is necessary to consider the issue that these individuals with very low fitness will make the start of optimization become very slow or stagnant in the fitness function. For example, the fitness score for individuals with the code 010 will be set to zero to ensure that the H3 requirements are fully met. However, because there are too many or all individuals in the

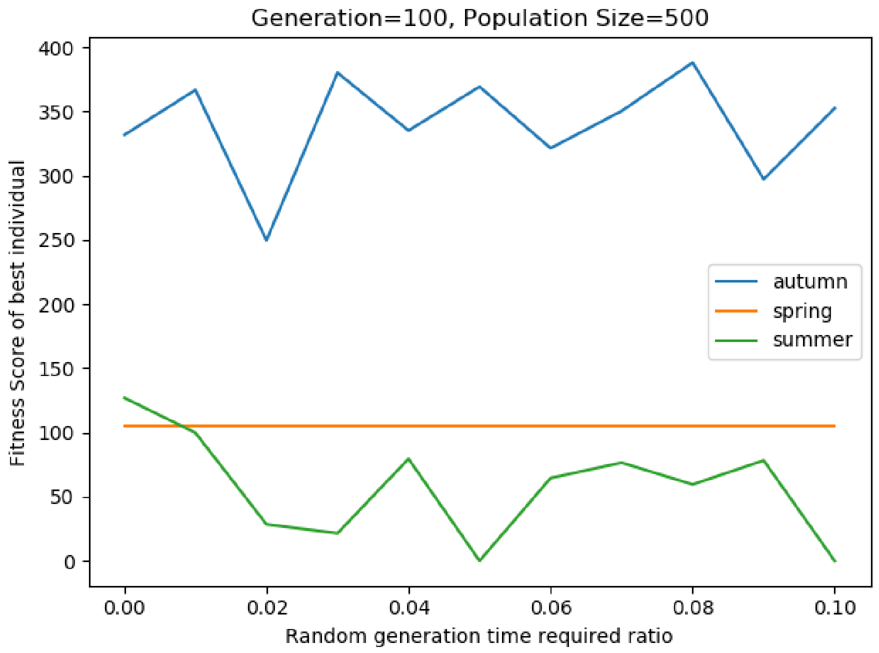


Figure 6.9: Fitness score curve (preference randomly generated from 0 to 100%)

initial population that contain the code '010' (After testing, everyone is highly likely to be included), so even in the case of iterations, the fitness of the population still won't improve. To solve this problem, the initialization of the population can be optimized. For instance, the pre-processing method can be used, because each line in the chromosome representation is a one-week schedule, and this schedule can be viewed as five one-day schedules. The one-day schedule is known from Chapter 4 in only seven cases. If the initial population is passed through a random five one-day schedules, then repeat N (student number) times. Then the obtained individual will necessarily satisfy H3. Such an initialized population already has a certain base number in the first fitness calculation, so that it will converge faster in the subsequent calculation and iterative process which means get the best fit score boundaries faster.

Secondly, in the process of population update, in order to ensure the size of the population is unchanged, the selection approach used in this project is the roulette selection method, and the subsequent crossover and variation are performed on the population as a whole according to a certain probability. Under such circumstances, the outstanding individuals of the previous generation may be completely replaced or changed. Therefore, in order to retain the outstanding individuals of the previous generation, only keep the excellent individuals when the selection operator is performed, and then the whole is crossed and mutated. Finally, the results of crossover and mutation are combined with the retained individuals, and as a new generation, the population size is kept constant. Such an approach can preserve the better individuals in each generation, and can also make the process of results faster.

In addition to the improvement of the algorithm, when considering the requirements of this project from a larger pattern, it can be considered whether the genetic algorithm is really suitable for solving this problem. As discussed in Chapter 5, when implementing this system, the use of genetic algorithms is limited in addressing the constraints of age-based priorities. Because each individual is generated as a whole during the process of initializing the population, the overall nature of the selection, crossover and mutation will not change. Therefore, such individuals cannot accurately generate desired results in an order based on age priority. Although the weight-based fitness function is used to solve this problem in the implementation process, as mentioned before, the random-based process does not allow the result to have complete order within. Because this project uses genetic algorithms to solve problems, no other approach is considered in the implementation process. However, in the process of thinking about solving problems, some methods that are not genetic algorithms have also been considered.

For example, the step of maintaining the initial population is unchanged, but when the individual, that is, the chromosome representation is crossed and mutated, the individual is not manipulated, but is operated sequentially according to the age-based priority. The schedule of the next student is optimized only if the line of high priority students meets the requirements. If this process gets stuck, go back to the previous step and reschedule the students at the previous level. In the end, all students get the right timetable. But the end condition of this method is not easy to define. If there is no suitable end condition, the backtracking process will either loop indefinitely or the results obtained will not be optimized.

In addition to the above method, it is also possible to use an individual (the entire timetable) as a unit, but a single student's timetable, that is, a timetable for generating one line at a time, so that the timetable in order of age-based priority can be generated. The timetable thus generated does not consider the vertical constraint H2 for the time being, so the vertical sum can be dynamically calculated in the process of generation. Once the generation schedule cannot satisfy H2, the schedule of the previous student is selected for rescheduling, if H2 is still not met, continue to go back to the previous one until the condition is met. Or until the number of iterations reaches a preset threshold.

In general, the two methods mentioned above have some drawbacks, but they all contribute to the processing of age-based priorities. None of these methods are genetic algorithms, or pure genetic algorithms. So when some problems are not solved perfectly using a single heuristic algorithm, a combination of methods can be used to achieve the goal.

6.5 Summary

This chapter first optimizes the four parameters in the genetic algorithm of the system. The robustness of the system was then evaluated using the optimized system. The results show that this system can solve most of the needs. At the same time, it also exposed some shortcomings. So I discussed some of the things that can be improved on the algorithm. Then jump out of the framework of the genetic algorithm, and propose a better possible solution to this problem.

Chapter 7

Conclusions

In this research work, the timetable system for the suitability of British child care institutions using genetic algorithms as the core was successfully completed. This is a very realistic project, because the requirements and input forms are realistic, and the data is based on facts. It also allows the end user to perform the actual test. Therefore, this project is quite challenging.

After learning that the project will use genetic algorithms as a solution, reading the literature on relevant heuristic algorithms and the literature using genetic algorithms to solve scheduling problems is the basis of the entire project. From these documents, some of the logic to solve the problem and the division and design of the specific modules ease the difficulty for the subsequent work. Then the analysis of the requirements and the enumeration of hard and soft constraints can give developers a very deep understanding of the project's goals. The end user can be put in place to consider the applicability and practicality of the system.

Next, combined with the reading of related literature, the whole system is divided into different modules and designed functions. By learning from previous experiences, such as using binary coding to simplify data manipulation, and innovative design based on specific problems, such as two-dimensional chromosome representations of timetables and population representations of three-dimensional arrays, the entire system is implemented in Python. Then through unit testing and integration testing and evaluation using randomly generated data. The entire system shows good applicability. Since the output data will be written directly into the spreadsheet of the input excel, it is read and understood by the end user. At this point, the objectives in the entire project have been basically achieved.

7.1 Suggestions for future work

As discussed in Chapter 6, this project has certain flaws in many respects due to time and method constraints. There are two situations in which improvements can be made to the project. The improvement can be focused on data pre-processing and elite retention strategies without changing the genetic algorithm as the core. If the genetic algorithm is not the only requirement, the backtracking method can be used to generate schedules in order of age-based priority and optimize in this order. In addition to the solution suggestions, Due to time constraints and limitations of the end user's computer operating environment, this project did not build a suitable user graphical interface. In the future, more complex editing of the results can be done in the spreadsheet file as an output, allowing end users to get more visual data from the spreadsheet.

Bibliography

[1] Aledo, J. A., Gámez, J. A., and Molina, D. Using metaheuristic algorithms for parameter estimation in generalized Mallows models. *Applied Soft Computing Journal 38* (2016), 308–320.

[2] Chen, C. F., Wu, M. C., Li, Y. H., Tai, P. H., and Chiou, C. W. A comparison of two chromosome representation schemes used in solving a family-based scheduling problem. *Robotics and Computer-Integrated Manufacturing 29*, 3 (2013), 21–30.

[3] Chen, J. F., Hsieh, H. N., and Do, Q. H. Predicting student academic performance: A comparison of two meta-heuristic algorithms inspired by cuckoo birds for training neural networks. *Algorithms 7*, 4 (2014), 538–553.

[4] Chen, Z., and Wang, R. L. Ant colony optimization with different crossover schemes for global optimization. *Cluster Computing 20*, 2 (2017), 1247–1257.

[5] Dai, M., Tang, D., Giret, A., Salido, M. A., and Li, W. D. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing 29*, 5 (2013), 418–429.

[6] Fang, H.-L., Ross, P., and Corne, D. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *PROCEEDINGS OF THE FIFTH INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS* (1993), Morgan Kaufmann, pp. 375–382.

[7] Hart, E., Ross, P., and Corne, D. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines 6*, 2 (2005), 191–220.

[8] He, Y., and Hui, C. W. A binary coding genetic algorithm for multi-purpose process scheduling: A case study. *Chemical Engineering Science 65*, 16 (2010), 4816–4828.

[9] Hu, N., and Fish, J. Enhanced ant colony optimization for multiscale problems. *Computational Mechanics 57*, 3 (2016), 447–463.

[10] John H. Holland. *Adaptation in Natural and Artificial Systems:An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press Cambridge, MA, USA 1992, 1992.

[11] Kirkpatrick, C. . D. . G., and Vecchi, M. . P. . Optimization by Simulated Annealing. *American Association for the Advancement of Science 220*, May (1983), 671–680.

[12] Marco, D., and Stützle, T. Ant Colony Optimization Theory. In *Ant Colony Optimization*. Elsevier Science Publishers Ltd. Essex, UK, 2004, pp. 121–152.

[13] Michael R. Garey and David S. Johnson. *Computers and intractability : a guide to the theory of NP-completeness*. W. H. Freeman, New York, 1979.

[14] Nakano, R., and Yamada, T. Conventional Genetic Algorithm for Job Shop Problems. *Proceedings of the 4th International Conference on Genetic Algorithms1*, JANUARY 1991 (1991), 474–479.

[15] Naseem, S., and Shengxiang, J. A Guided Search Genetic Algorithm for the University Course Timetabling Problem. *Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2009)* (August 2009), 10–12.

[16] Shen, M., Zhan, Z.-H., Chen, W.-N., Gong, Y.-J., Zhang, J., and Li, Y. Bi-Velocity Discrete Particle Swarm Optimization and Its Application to Multicast Routing Problem in Communication Networks. *IEEE Transactions on Industrial Electronics 61*, 12 (2014), 7141–7151.

[17] Shen, M., Zhan, Z.-H., Chen, W.-N., Gong, Y.-J., Zhang, J., and Li, Y. Bi-Velocity Discrete Particle Swarm Optimization and Its Application to Multicast Routing Problem in Communication Networks. *IEEE Transactions on Industrial Electronics 61*, 12 (2014), 7141–7151.

[18] Wang, S., Xiao, C., Liu, W., and Casseau, E. A comparison of heuristic algorithms for custom instruction selection. *Microprocessors and Microsystems 45* (2016), 176–186.

[19] Zhan, Z.-H., Member, S., Zhang, J., Member, S., Li, Y., and Shi, Y.-H. Orthogonal Learning Particle Swarm Optimization. *Ieee Transactions on Evolutionary Computation*, November (2010).

Appendices

Unit test:

```
__author__ = 'Sheng_XU'

import unittest
import numpy as np

case1 = np.array([[0, 1, 1, 1, 0, 1, 0, 1, 1],
                  [1, 1, 1, 0, 0, 1, 1, 0, 1],
                  [0, 0, 1, 1, 0, 1, 1, 1, 0],
                  [0, 1, 1, 0, 0, 0, 0, 1, 1],
                  [1, 1, 0, 0, 1, 1, 1, 0, 1]])

case1_fitness_score = 63.5

case2 = np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0]])

case3 = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1],
                  [1, 1, 1, 1, 1, 1, 1, 1, 1],
                  [1, 1, 1, 1, 1, 1, 1, 1, 1],
                  [1, 1, 1, 1, 1, 1, 1, 1, 1],
                  [1, 1, 1, 1, 1, 1, 1, 1, 1]])

case4 = np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [1, 1, 1, 1, 1, 1, 1, 1, 1],
                  [1, 1, 1, 1, 1, 1, 1, 1, 1],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [1, 1, 1, 1, 1, 1, 1, 1, 1]])

case5 = np.array([[0, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 1, 0, 0, 0, 0, 0]])


def fitness(table, constrains, col_constrain):
    for i in range(5):
        line = table[i]
        totalhours = 0
        perference = []
        timeslots = np.split(line, 5)
        for num in range(5):
            ary = timeslots[num]
            hours = cal_timeslots(ary)
            if hours != -1:
                totalhours += hours
            else:
                totalhours += 1.25
                if result - 10 > 0:
                    result -= 10
                else:
                    result = 0
            perference.append(10 * (num + 1) + cal_perference(ary))
        if constrains[i][2] != 0:
            if constrains[i][2] in perference:
                result += constrains[i][5]
        else:
            result += 1
        if totalhours == constrains[i][4]:
            result += 20
        elif totalhours < constrains[i][4]:
            result += 1
        else:
            if result - 5 > 0:
                result -= 5
            else:
                result = 0
    p = 0
    for col in table.T:
        if np.sum(col) + col_constrain[p] > 26:
            result = 0
        p = p + 1
    return result

# calculate the binary number like 101
def cal_perference(ary):
    bin2int = convertArrary(ary)
    if bin2int == 1 or bin2int == 3:
        result = 2
    elif bin2int == 6 or bin2int == 4:
        result = 1
    else:
        result = 0
    return result

# calculate the total hours of each child by
# the binary number like 101
def cal_timeslots(ary):
    bin2int = convertArrary(ary)
    if bin2int == 0:
        result = 0
    elif bin2int == 1 or bin2int == 4:
        result = 2.5
    elif bin2int == 3 or bin2int == 6:
        result = 3.75
    elif bin2int == 5:
        result = 5
    elif bin2int == 7:
        result = 6.25
    else:
        result = -1
    return result

def convertArrary(ary):
    string = ''
    for num in ary:
        string += str(num)
    bin2int = int(string, 2)
    return bin2int


def crossover(parent1, parent2, cross_rate):
    if np.random.rand() < cross_rate:
        random_individual = parent2
        points = [1, 3, 5]
        parent1[points]=parent2[random_individual,points]
    return parent1
```

```
def mutate(child, mutate_rate):
    if np.random.rand() < mutate_rate:
        child.T[3] = abs(child.T[3] - 1)
    return child

class TestGA(unittest.TestCase):

    def test_fitness(self):
        self.assertEquals(fitness(case1, constrians,
         \\ col_constrains), case1_fitness_score)

    def test_convertArrary(self):
        self.assertEquals(convertArrary(np.array
        \\ ([1, 0, 1, 0, 1, 1])), '101011')

    def test_cal_perference(self):
        self.assertEquals(cal_perference(np.array([1, 0, 1])), 0)

    def test_cal_timeslots(self):
        self.assertEquals(cal_timeslots(np.array([1, 0, 1])), 5)

    def test_crossover(self):
        self.assertEquals(crossover(case2, case3), case4)

    def test_mutate(self):
        self.assertEquals(mutate(case2), case5)

if __name__ == '__main__':
    unittest.main()
```