

A Hierarchical Progressive Genetic Algorithm-Based Course Scheduling System for Curricular-Variable System

Luofu Gong

*Beijing Aidi School, Beijing, China
13359226905@163.com*

Abstract. This paper addresses the problem of automatic course scheduling, a core requirement in educational institutions and a typical NP-complete combinatorial optimisation problem. The complexity of this problem is significantly increased in real-world scenarios due to the coexistence of multiple constraints, including cross-module courses, parallel courses, consecutive period requirements, teacher availability, and classroom restrictions. These factors make manual scheduling extremely time-consuming and render classical search algorithms such as brute-force enumeration and backtracking impractical within a reasonable time frame. To tackle this challenge, a Hierarchical Progressive Genetic Algorithm (HPGA) is proposed. The algorithm adopts a multi-stage optimisation strategy, where combinatorial courses are first evolved to generate a feasible schedule template, followed by the insertion and evolution of standalone courses to construct a complete schedule. To preserve inter-course dependencies during genetic operations, a encoding scheme is , enabling compatibility with Partially Mapped Crossover (PMX). Experimental results based on real-world high school data demonstrate that the proposed system can consistently generate penalty-free schedules with high efficiency, achieving an average execution time of 0.7781 seconds. Compared with traditional genetic algorithms, HPGA shows improved feasibility, scalability, and robustness in handling complex scheduling scenarios.

Keywords: Course Scheduling, Genetic Algorithm, Combinatorial Optimisation, HPGA

1. Introduction

Automatic course scheduling presents a task of both educational necessity and practical challenge. As one of the most crucial and core demands in educational administration in any educational institution, schedule arrangement ensures the orderly conduct of school teaching activities, directly impacting the instructional efficiency, resource utilisation, and the experience of both teachers and students. Manual scheduling suffers from several limitations. Its success relies on the experience of the administrative staff, often requiring weeks of dedication while still being prone to unconscious errors. Moreover, during the scheduling process, it may be discovered that certain course combinations selected by some students are fundamentally impossible to accommodate, necessitating a complete restart and consuming a large amount of additional time. With the completion and complication of the curriculum system along with the expansion of school size and

reformation of the teaching system, it is profoundly realised that the automation of course scheduling is becoming an urgent priority for educational administration.

Course scheduling is a classical NP-complete combinatorial optimisation problem [1]. Its solution space grows rapidly with the number of variables involved [2]. Beyond the basic structure of a single course, which involves one course at one time, with one teacher in one room, the real-world scenarios are far more complex, with multiple sorts of constraints, including course format, temporal requirements, and resource limitations. To illustrate the various forms of courses with examples, course formats may include cross-module courses (where students from different modules take the same course simultaneously with the same teacher) and parallel courses (where parts of a module take different courses at the same time), or a combination thereof. Temporal constraints also require careful handling, such as the need for consecutive class periods, courses that run on alternate weeks, and teachers' fixed unavailability period. Furthermore, resource limitations imposed by the finite availability of teachers and classrooms must also be considered.

Although some other algorithms demonstrate a certain degree of feasibility to solve related problems, each of them also has significant limitations that cannot be ignored. Exact algorithms such as enumeration and backtracking are theoretically capable to obtain an precise solution; however, their calculation complexity is prohibitively high [3], which confines their applicability to extremely small-scale problems, excluding their use in practical scenarios [4]. Traditional heuristic algorithms [5], apart from genetic algorithm such as simulated annealing [6] and taboo search [7], tend to converge to local optima [8] and struggle with multi-peak, highly constrained search spaces [9]. Furthermore, their performance is often highly sensitive to parameter settings [10], resulting in inadequate robustness. Although the standard genetic algorithm possesses global searching ability [11], it nevertheless suffers from slow convergence and premature convergence [12] when dealing with high-density constraints. Additionally, traditional encoding schemes often fail to preserve inter-course relationships [13] such as synchronisation of cross-module courses, leading to a high proportion of invalid solutions.

Current methods often struggle to balance feasibility, efficiency, and solution quality when confronted with the multi-level, strongly coupled constraints of real-world environments [14]. To bridge the gap, this paper proposes the Hierarchical Progressive Genetic Algorithm (HPGA), which is an enhanced genetic algorithm designed specifically for complex educational scenarios.

1.1. Definition of terms

The following are explanations of special terms in complex educational schedules, which will be used further in this paper.

(1) Module: A distinct group of students for different specific educational programs. Each student is assigned to exactly one module. Each module contains students from exactly one grade.

(2) Course: A scheduled teaching session, uniquely defined by its assigned teacher, classroom, and involved module(s). It occupies a specified number of periods in the schedule.

(3) Standalone Course: A course taken by exactly one complete module.

(4) Cross-Module Courses: Courses that are taken by students from more than one module simultaneously with the same teacher and the same classroom

(5) Parallel Courses: A set of courses simultaneously taken by different students from the same module.

(6) Consecutive Period Courses: Courses that requires to occupy certain number of consecutive periods.

(7) Biweekly Courses: Courses that take place in alternative weeks.

(8) Combinatorial courses: A set of different courses of different modules that are bundled together and must be taken concurrently in the same period.

The aforementioned concepts and requirements are predetermined based on administrative needs. They are provided as fixed input and constraints to the scheduling system.

1.2. Overview of curriculum model

Module-A						Module-B					
	Mon	Tues	Wed	Thur	Fri		Mon	Tues	Wed	Thur	Fri
1 st	MATH	MATH	ENG-C	MATH	CHEM	1 st	MATH	MATH	ENG-F	MATH	CHEM
2 nd	MATH	MATH	MATH	CS/BIO(A)/ ECON	CHEM	2 nd	MATH	MATH	MATH	CS/BIO(A)/ ECON	CHEM
3 rd	CS/BIO(A)/ ECON	CHEM	CS/BIO(A)/ ECON	ENG-C	PSY (even week)	3 rd	CS/BIO(A)/ ECON	CHEM	CS/BIO(A)/ ECON	ENG-F	PSY (even week)
4 th	CS/BIO(A)/ ECON	ENG-F	MATH	CHEM	MATH	4 th	CS/BIO(A)/ ECON	ENG-C	MATH	CHEM	MATH
5 th	PHE	CS/BIO(A)/ ECON	PHY/BIO(B)	PHY/BIO(B)	CS/BIO(A)/ ECON	5 th	PHE	CS/BIO(A)/ ECON	PHY/BIO(B)	PHY/BIO(B)	CS/BIO(A)/ ECON
6 th	PHE	CS/BIO(A)/ ECON	PHY/BIO(B)	PHY/BIO(B)	MATH	6 th	PHE	CS/BIO(A)/ ECON	PHY/BIO(B)	PHY/BIO(B)	MATH
7 th	PHY/BIO(B)	CHEM	CHEM	ENG-F	Counselling	7 th	PHY/BIO(B)	CHEM	CHEM	ENG-C	Counselling
8 th	PHY/BIO(B)	PHY/BIO(B)	CHEM	CHEM	ENG-F	8 th	PHY/BIO(B)	PHY/BIO(B)	CHEM	CHEM	ENG-F

Figure 1. An example of the current schedule of two modules in a complex educational scenario

As shown above in Figure 1, these two modules, from the same grade, are perfect examples to illustrate the curriculum model in certain complex educational scenarios. Take the example of the first period on Monday, the math courses in both modules are combinatorial courses; they are taken in different classrooms with different teachers, but must be taken simultaneously. The 5th and 6th courses on Monday are examples of cross-module courses, as PHE courses are considered to be the same course taken by all modules during specified periods. The 4th course on Tuesday represents a single course, as it is taken by one module. Meanwhile, the 3rd and 4th courses on Monday reveal the most complicated situation: it is a combination of a cross-module course and parallel courses. CS, BIO(A), and ECON are cross-module courses, while they are also parallel courses for each individual module. What is even more complicated is that CS is taken across module B and C, BIO(A) is taken by modules A, B, and C, while ECON is taken by module B, C and D. In summary, the main reason for the complexity is that the courses within the combination of cross-module courses and parallel courses involve different modules, as shown in Figure 2 below.

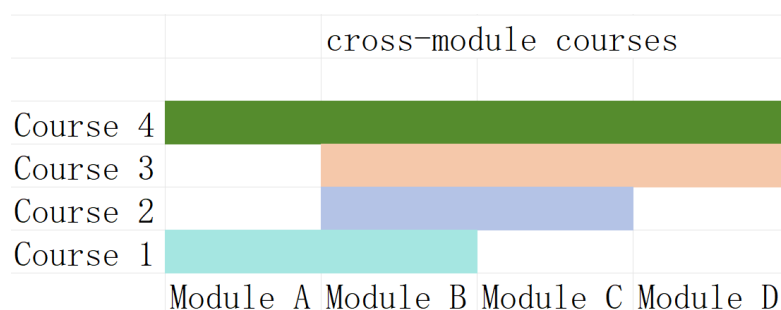


Figure 2. Cross-module courses with varying lengths

Furthermore, the 3rd and 4th courses on Monday are also consecutive courses, the same as the 7th and 8th courses on Monday, and the 3rd course on Friday is a case in point of biweekly courses.

1.3. Research challenges and design process

To construct a practical, robust system capable of satisfying various constraints while maintaining good convergence, we went through three crucial design iterations. Each failure revealed the limitations of traditional methods and guided subsequent improvement.

Traditional sequential encoding was first adopted. In this approach, a group of parallel courses was treated as a single course for the corresponding module, and cross-module courses were manually aligned during the generation of the initial population. Each course for a separate module was considered an independent gene for crossover and mutation. However, to maintain a constant course count, we introduced Partially Mapped Crossover (PMX), which created a dilemma: if courses within a cross-module group were treated as independent genes, PMX would break their synchronisation and disrupt teachers' fixed teaching loads; conversely, if their internal dependencies were considered, numerous invalid parallel courses would be generated.

	Before PMX			After PMX	
Period 1	Course 1	Course 2		Period 1	Course 1/3 Course 3
Period 2	Course 3	Course 3		Period 2	Course 3 Course 3
	Module A	Module B			Module A Module B

Figure 3. Formation of invalid parallel courses

Figure 3 demonstrates the formation of invalid parallel courses. Assume Course 2 and Course 3 are mapped during crossover. To substitute Course 2 with Course 3, another Course 3 would be synchronously added to Module A in Period 1, since Course 3 is a cross-module course involving both Module A and Module B. However, it is then noticed that Course 1 is originally assigned to Module A in Period 1. This produced an invalid parallel course in Module A, as students would inevitably overlap between Course 1 and Course 3. This trend prevented a penalty-free result from being generated. The fundamental reason is the incompatibility of PMX crossover with traditional sequential encoding when dealing with genes that have internal dependencies, revealing that encoding must embed inter-course constraints.

To address this dependency, the second model splits each module into sub-modules, breaking the parallel course relationship. As each student belongs to exactly one sub-module, parallel course constraints could be ignored, allowing the system to focus solely on cross-module courses. A penalty based on total teaching periods was introduced. The algorithm was expected to align cross-module courses.

Initial experiments on three sub-modules of one module met expectations. However, when applied to an entire grade (about eight sub-modules), the system failed to converge before the population stagnated (zero variance). Attempts to resolve this, including backtracking, also failed. The root cause was that ignoring parallel courses shifted the burden to aligning sub-module courses originally belonging to the same full module. This approach broadened the search space immensely, and the penalty alone could not guide targeted search. This model revealed that course scheduling requirements cannot be simplified by complete decoupling, and an efficient search strategy capable of jointly handling constraints between courses and modules must be devised.

In conclusion, it was precisely these defects that gave rise to HPGA, leading to the ultimate solution for this complex scheduling scenario.

2. Methods

2.1. Algorithm workflow

Although the algorithm is indeed designed aiming for progressive scheduling of multiple grades, which can be identified as a module combination with no combinatorial courses across different grades unless specific occasions that are urgently prioritised, the workflow illustrated focuses on the complete progression of the algorithm taken on each grade, as the operations taken on each grade are of the same process.

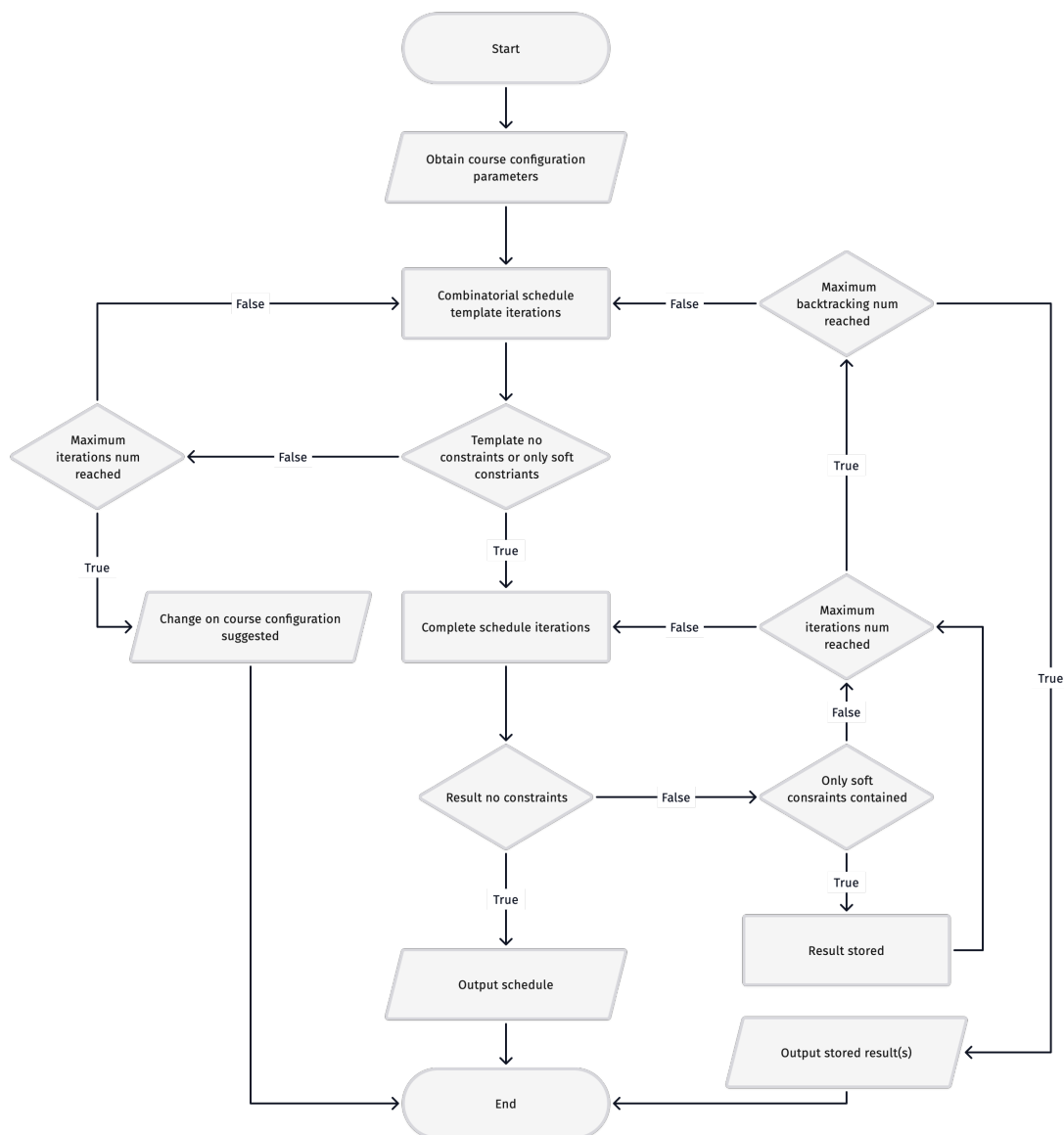


Figure 4. Algorithm workflow flowchart

As demonstrated in the Figure 4 flowchart above, after the course configuration information is obtained, the combinatorial schedule template is first attempted to generate through GA iteration: initialisation, repeated crossover and mutation. It is judged whether the final outcome is free of

constraints or contains only soft constraints. The re-iteration on combinatorial schedule templates takes place if hard constraints are still present in the result, up to a limited amount of time. A message would be sent to the user to suggest an adjustment on course configuration if there are still hard constraints remaining after the maximum number of repeating reached.

The iterations on the complete schedule starts since a constraint-free or soft constraints only combinatorial schedule template is successfully generated. Selection on whether the complete schedule result is constraint-free takes place, where constraint-free schedules are directly considered as the final result as well. Distinctively, the result containing only soft constraints is not directly output as the final result, but rather stored, and a backtracking to the regeneration of the combinatorial schedule template takes place unless the maximum backtracking amount is reached. If a constraint-free result is ultimately generated, it is outputted; otherwise, the stored results are outputted, with each clarified as to its remaining constraints.

2.2. Chromosome encoding

As mentioned in Section 1.4, sequential encoding was the first choice for the scheduling system because the individual genes within a chromosome of the schedule are discrete and interdependent, and constraints such as fixed course counts and consecutive periods require the adoption of sequential encoding. However, traditional sequential encoding lacks compatibility with PMX crossover when handling inter-course dependencies. To address the challenge, a modified encoding is adopted in this system.

The overall structure of a chromosome is a three-dimensional nested list. The first dimension represents time periods, with a fixed length of *total_periods*, representing the total number of periods in each week. The second dimension represents modules, and the third dimension is a list of course entities assigned to that module in that period (the length of this list may vary).

To formally describe the encoding scheme, we introduce the following notation:

Individual courses are denoted as

$$C_n^f = \{Tchr, Clrm, Miv\} \quad (1)$$

For which, *f* represents a flag that can be either empty as C_n or value as a * to be C_n^* , denoting standalone course and combinatorial course respectively. The subscript *n* of represents the index of the current course located in the corresponding course list, distinguished as standalone course list *SCL* and combinatorial course list *CCL*, denoted as

$$SCL = \{C_1, \dots, C_k\} \quad (2)$$

$$CCL = \{C_1^*, \dots, C_n^*\} \quad (3)$$

Such that, *n* of each type of course do not surpass the length of the corresponding course list.

Tchr, *Clrm*, and *Miv* represent the teacher of the course, the classroom of the course, and the module involved in this course. For standalone courses, only one module is included in the *Miv*,

whereas multiple modules are involved in the combinatorial course.

For T and M represent periods and modules, such that

$$T = \{1 \dots \text{total_periods}\} \quad (4)$$

$$M = \{1 \dots \text{total_modules}\} \quad (5)$$

The gene G located at a specific chromosomal location can be defined as

$$G_{t,m} \leftarrow C_n^f, t \in T, m \in M, C_n^f \in SCL \cup CCL \quad (6)$$

Each gene takes a value of a certain course, and each course can be repeatedly assigned to different genes.

Conclusively, the chromosome S can be represented as

$$S = \{G_{t,m} \mid t \in T, m \in M\} \quad (7)$$

Further more, let $M(C_n^*)$ denotes the Miv of a combinatorial course, for a combinatorial course C_n^* , the inter-course synchronisation constraint can be denoted as

$$\forall C_n^* \in CCL, \forall t \in T: (\exists m \in M(C_n^*), G_{t,m} \leftarrow C_n^*) \implies (\forall m' \in M(C_n^*), G_{t,m'} \leftarrow C_n^*) \quad (8)$$

Such that, for all combinatorial courses and all periods, under the condition that a combinatorial course C_n^* is assigned to $G_{t,m}$ where m belongs to the Miv of this combinatorial course, it must be ensured that the same combinatorial course must be assigned simultaneously to genes of each individual module m' that as well belongs to the Miv of the same combinatorial courses at the same period.

To clearer explain this constraint, a hypothetical combinatorial course C_1^* is defined that it involves both module A and B. As shown in Figure 5, the chromosome is valide if and only if C_1^* exists in both module A and module B; if C_1^* exist in only module A, the chromosome is invalid; on the other hand if C_1^* exists in any other module other than module A and B, the chromosome is invalid as well. The rule applies to all combinatorial courses at any period.

	Module-A	Module-B	Module-C	Module-D	Module-E	
Period 1	C_1^*	C_1^*				Valid
Period 2	C_1^*					Invalid
Period 2	C_1^*	C_1^*		C_1^*		Invalid

Figure 5. Inter-course synchronisation

This separate labelling mechanism enables compatibility between sequential encoding and PMX crossover by distinguishing genes that have an internal connection from those that have none, thereby allowing hierarchical genetic operations.

2.3. Fitness calculation

In this system, the fitness function not only serves to select and filter high-quality individuals, but also guides the direction of evolution. Since not all penalty items are of equal importance, different weight coefficients are assigned to each, and the fitness function adopts the weighted sum of all penalty terms [15], expressed as:

$$\text{fitness} = \sum_{k=1}^n W_k \times ST_k \quad (9)$$

For which W_k represents the weight of each constraint, T represent the number of times this constraint is triggered, and n is the total number of constraints. The fitness function is therefore cumulative; the higher the fitness, the worse the individual.

Constraints are divided into two categories: hard constraints and soft constraints [16]. Hard constraints are fundamental requirements that must be satisfied; violating them renders the schedule completely infeasible. Soft constraints are not strictly required but reflect the rationality and quality of the schedule.

Based on the educational requirements of the target scenario, nine penalty items are defined, as listed in Table 1. The specific weights value mentioned below are decided by discriminating hard and soft constraints by assigning hard constraints domination positions, while remaining the influence of soft constraints. The range of weights of each constraint is suggested due to multiple adjustments to weight values.

Table 1. Penalty items

Constraints	Type	Weight	Description
Teacher conflicts	Hard	5-10	A teacher is assigned to more than one different courses during the same period.
Classroom conflicts	Hard	5-10	A classroom is assigned to more than one course during the same period.
Consecutive periods	Hard	5-10	Required consecutive periods are not satisfied.
Teacher unavailability	Hard	5-10	A course is scheduled in a teacher's prohibited time period.
Classroom unavailability	Hard	5-10	A course is scheduled in a classroom's prohibited time period.
Module occupation conflict	Hard	5-10	A course is scheduled in a module that is already occupied by another combinatorial course in the same period.
Empty set / Self-study penalty	Soft	1-2	Empty course / self-study course locating at improper position
Excessive daily courses	Soft	0.1-0.5	The same course appears in a module for more than twice a single day.

2.4. Population iteration

In genetic algorithms, both the Generational Model [17] and the Steady-State Model are classical population models [18]. This system adopts the Steady-State Model due to its advantages over the Generational Model. The Generational Model replaces all parents with newly generated offspring. This strategy tends to lose excellent genes and suffers from low convergence stability due to oscillations that occur during the evolutionary process [19]. In contrast, the Steady-State Model shows better convergence, as the top 50% of individuals are retained. This strategy demonstrates its advantages in retaining elite individuals continuously, resulting in a smooth convergence curve and avoiding degradation of the optimal solution [20].

For the selection strategy, traditional Roulette Selection has the limitation of losing diversity, as super individuals would rapidly dominate the population when fitness differences are too large [21]. Therefore, Rank-based Roulette Selection was adopted [22]. This selection tactic first sorts the individuals by fitness in ascending order. Probabilities are assigned to each individual based on its rank, rather than on raw fitness values or uniformly. The selection formula is presented as:

$$P(r) = \frac{N-r}{\sum_{k=1}^N k} = \frac{2(N-r)}{N(N+1)} \quad (10)$$

Where $P(r)$ denotes the probability of individuals at rank r being selected, and N denotes the total number of individuals in the population.

To illustrate the population updating process, we denote the parent populations at the t^{th} iteration as P_t with constant size N . Crossover and mutation operations are performed by half of N times, creating exactly the same number of population individuals as the original population (where N has to be an even number), each time selecting two individuals from the parent population, to generate the descendant population denoted as Q_t . Then the union $P_t \cup Q_t$ is formed, resulting in a combined population of size $2N$ as an intermediate state. This intermediate population is then sorted

by fitness; the first N individuals are selected as the new generation P_{t+1} . The iteration counter t is then incremented.

As the schedule is generated hierarchically, all operations are performed sequentially for each grade. Iterations are performed separately for each grade, as mentioned above. Three termination conditions are defined: reaching the maximum number of iterations, successfully generating a feasible solution (fitness = 0), and detecting premature convergence (stagnation). If the fitness of the best individual in the current group reaches zero, the program proceeds to schedule the next grade. Failure in scheduling the current grade is indicated by two conditions:

(1) The current iteration count has exceeded the maximum number of generations, and the best individual's fitness has not reached zero.

(2) The number of consecutive iterations with no improvement (i.e., the population fitness variance diminishes while the best fitness remains unchanged) has reached a predefined maximum stuck count.

In such cases, the algorithm backtracks to the previous data set that the current data set depends on to regenerate a different feasible result.

2.5. Combinatorial schedule template

The combinatorial schedule template is at the very first hierarchy of the system, takes the responsibility of lessen searching space for the complete schedule, to enhance the efficiency of both combinatorial and complete schedule iterations, while avoiding invalid chromosome been produced.

Although it is acceptable that there are remaining soft constraints within the combinatorial schedule template, a constraint-free template is nevertheless rather expected to be produced, in order to guarantee the effectiveness of the generated complete schedule. Thus, numbers of iterations should be taken in order to generate a penalty-free template if the first generation failed to reach the expectation.

The core principle of the following operations on the combinatorial schedule is that no more than one combinatorial course may occupy any given period in the current schedule, which fits the requirement. This regulation ensures the feasibility of crossover in combinatorial schedules.

2.5.1. Initialisation of combinatorial schedules

As the core principle illustrated in Section 2.5, the chromosome can be compressed as a one-dimensional array of fixed length of *total_periods*, filled with either a combinatorial course or an empty placeholder at each position.

As shown in Figure 6, each period of the schedule is extracted and compressed by only remaining the combinatorial course if one exists, otherwise a blank is left.

Such a design not only ensures the validity of the chromosome through the following operations, but also simplifies the operations while maintaining the inter-modular synchronisation as mentioned in Section 2.2.

It has to be taken into consideration that some special combinatorial courses are manually assigned to some specific periods in advance, which are considered as a superior hierarchy. Those of the superior hierarchy must be set to the fixed position in advance for every combinatorial schedule of individual chromosomes during the initialisation process, and prevent any alteration from being performed on them.

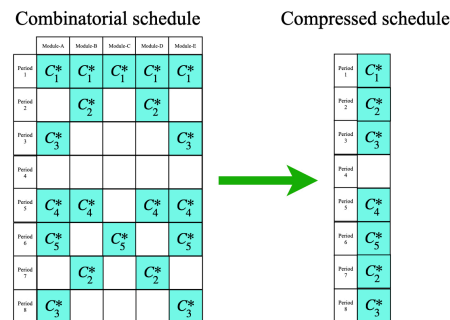


Figure 6. Combinatorial schedule compression

2.5.2. Crossover for combinatorial schedules

The crossover of combinatorial schedules provide possible enhancement on chromosome positional relationship while ensuring the total period count of each course remains constant. The partially mapped crossover (PMX) algorithm is applied to reach the expected effect.

An exchange range of random length is first decided, and a random start point is generated with the prevention of index out of bounds. The indices of each combinatorial course are found in advance. Two descendant chromosomes corresponding to one of the parents are produced by adding the gene from the corresponding parent at each period. The amount of time which a combinatorial course has existed more than required is counted, for which those with excessive periods will be deleted from outside the exchange range, and substituted by those with deficient periods.

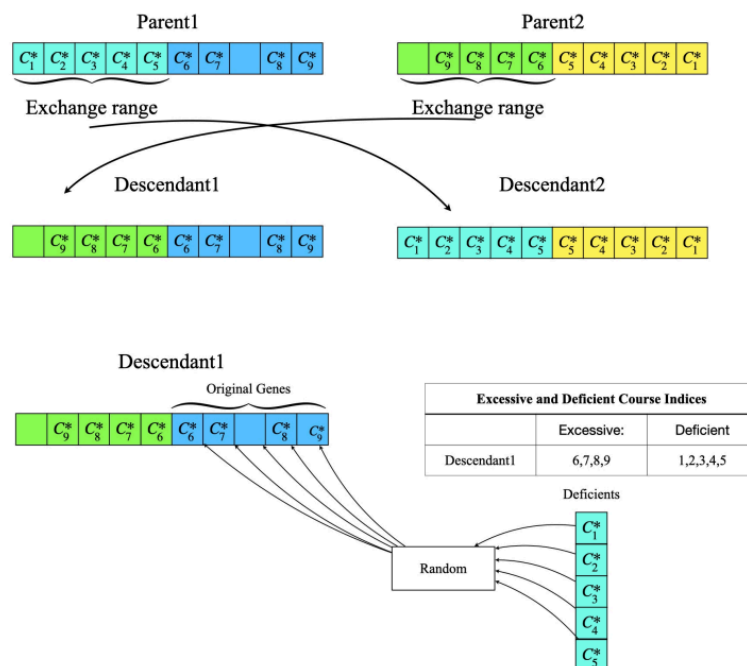


Figure 7. Combinatorial schedule crossover operation

As shown in Figure 7, the chart (1) illustrates the process of direct exchange, where the first five gene positions are chosen as exchange range. The same coloured gene fragments are the same. Take example of descendant1 in diagram (2), it is observed that combinatorial course 6-10 have one more period each than the original requirement, while combinatorial course 1-5 occupied none of the periods, which is one less than the requirement. The deficient genes are extracted from the course list and replace an excessive gene with random order.

The progress above ensures the invariability of the course count of each combinatorial course, and the effectiveness of the exchange as the exchanged fragment of gene is stay unchanged.

2.5.3. Mutation for combinatorial schedules

The mutation operation aims to strengthen the contextual relationships within the schedule by period relocation, thereby reducing penalties related to consecutive periods and excessive daily courses.

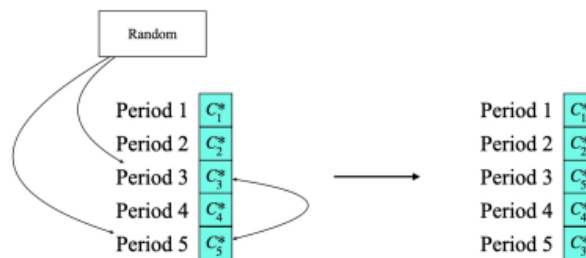


Figure 8. Mutation operation

As demonstrated in Figure 8, two distinct indexes are randomly selected, and the genes at these two positions are then swapped.

To avoid the premature convergence observed in previous models, a constant mutation probability is adopted for all individuals, rather than a dynamic one. This ensures that individuals stuck in local optima still have opportunities for adjustment.

2.5.4. Template formation

The iteration stops when the fitness of the best individual in the current population reaches zero. The best individual is then considered as a template for the following complete schedules. It is first converted into a full three-dimensional schedule using the structure described in Section 2.2.

Since the feasibility of the current state depends on previous schedules and templates, if the generation of a penalty-free individual fails, the process of current template generation is repeated up to a certain number of times. If it still cannot produce the expected result, as mentioned in Section 2.1, remaining soft constraints would be adopted for the current stage.

2.6. Complete schedule generation

Following the generation of combinatorial schedule templates, the progressive schedule creation involves filling in standalone courses to produce a complete schedule. The standalone courses, along with the self-study courses, are considered a second hierarchy with lower deposit priority.

Based on the template generated, the overall purpose of the current process is to fill all standalone courses into the schedule, ultimately generating a penalty-free and feasible grade schedule that is perfectly compatible with the previous generation.

The initialisation of complete schedules can generally be considered similar to that of courses. Firstly, the standalone courses, along with a certain amount of self-studies, are initially selected. The schedule template is copied to serve as the foundation of the current complete schedule chromosome. The selected standalone and self-study courses are shuffled and filled into the template to generate a complete schedule.

During the process, it must be aware that there must never be a standalone course or self-study course placed in the same position as a combinatorial course, to ensure the validity of the chromosome.

During the crossover of the complete schedule, only the courses of the second hierarchy, which are the standalone courses, are considered in the operation, whereas the courses of the first and superior hierarchy are unmoved throughout the process.

A similar process as described in Section 2.5.2 is taken on all the standalone courses, resulting in a partial complete exchange and shuffle of original gene to a certain extent.

As shown in Figure 9, the above figure demonstrates the gene exchange process of complete schedules, where standalone courses located in the period that is included in the exchange range is switched, while the combinatorial courses are remained. In chart below otherwise, since same processes are taken on each module courses, the module-A courses of descendant1, it is seen that standalone course 3 is excessive by one period and standalone courses 2 is deficient by one. The deficient course then replaced a random excessive course that is not located in exchange range, thus remains the validity of chromosome. A similar operation as described in Section 2.6.3 is taken on the second hierarchy course.

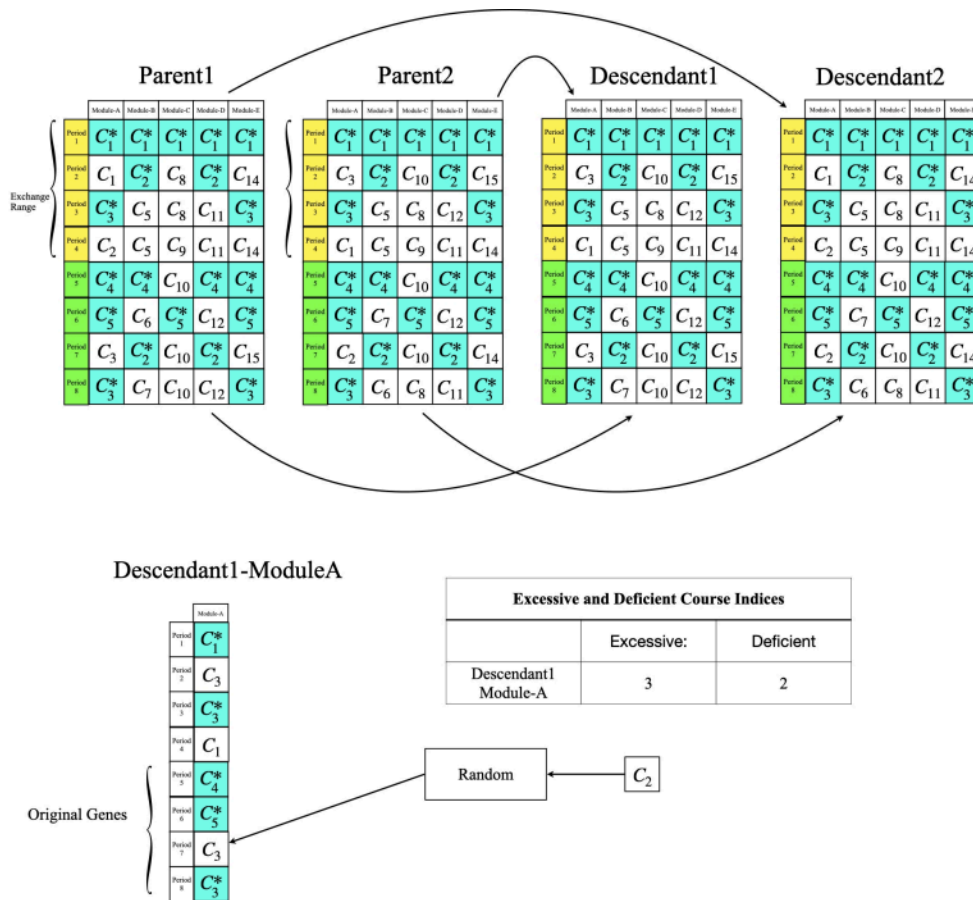


Figure 9. Complete schedule crossover operation

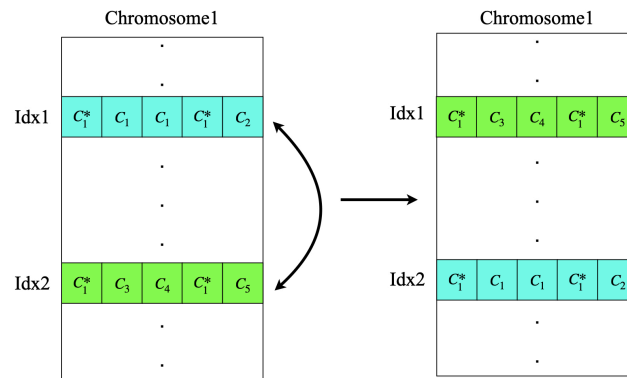


Figure 10. Complete schedule mutation operation

As shown in Figure 10, the indexes of exchange are no longer selected randomly, but those with the same combinatorial course are selected. The reason for that is to retain the invariant of the schedule template in each chromosome, by exchanging the same combinatorial courses along with the different standalone courses, effectively improving species variety while assuring the position of combinatorial courses stays constant.

3. Result and discussion

Current data from a high school with a complex educational scenario and requirements is selected for testing. The examination is taken on Apple M4 CPU, 16GB internal storage, with Python version 3.14.

	Module A	Module B	Module C	Module D	Module E	Module F	Module G	Module H
1	CHEM	MATH	CHI	PHY	PHY/BIO	BIO/PHY	PHY	ENG-C
2	MATH	PHY	ACC	CHEM	MATH	MATH	Self-Study	BUSS
3	PHY	CHEM	ACC	BIO/ECON/CS	MATH	MATH	MATH	Self-Study
4	ENG	ENG	ENG	ENG	ENG-F	COUN	MATH	BUSS
5	PHE	PHE	PHE	PHE	PHE	PHE	PHE	PHE
6	PHE	PHE	PHE	PHE	PHE	PHE	PHE	PHE
7	ECON/CS/BIO	ECON/BIO/CS	MATH	CHI	PHY/BIO	BIO/PHY	ENG-F	MATH
8	ECON/CS/BIO	ECON/BIO/CS	ACC	CHEM	CHEM	CHEM	CHEM	Self-Study
9	CHEM	PHY	PSY	CHI	MATH	MATH	MATH	BUSS
10	ENG	ENG	ENG	x	PHY/BIO	BIO/PHY	MATH	COUN
11	PHY	CHEM	BUSS	CHI	PHY/BIO	BIO/PHY	ENG-C	Self-Study
12	PHY	CHEM	MATH	BIO/ECON/CS	PSY	ENG-C	COUN	ENG-F
13	MATH	PHY	BUSS	CHEM	CS/ECON	ECON	BIO/ECON/Extra-ENG	ECON/Extra-ENG/ACC
14	CHI	Self-Study	CHI	MATH	CHEM	CHEM	CHEM	MATH
15	CHEM	MATH	Self-Study	BIO	CS/ECON	ECON	BIO/ECON/Extra-ENG	ECON/Extra-ENG/ACC
16	CHEM	MATH	BUSS	MATH	CS/ECON	ECON	BIO/ECON	ECON/ACC
17	CHEM	MATH	Self-Study	CHEM	MATH	MATH	ENG-C	PSY
18	CHI	CHEM	MATH	CHEM	CS/ECON	ECON	BIO/ECON/Extra-ENG	ECON/Extra-ENG/ACC
19	ECON/CS/BIO	ECON/BIO/CS	Self-Study	PHY	PHY/BIO	BIO/PHY	PHY	BUSS
20	ENG	ENG	ENG	ENG	ENG-F	ENG-F	PHY	MATH
21	Self-Study	CHI	Self-Study	BIO/ECON/CS	CHEM	CHEM	CHEM	MATH
22	PHY	COUN	MATH	BIO/ECON/CS	CHEM	CHEM	CHEM	ENG-C
23	BIO	CHI	BUSS	PHY	MATH	MATH	ENG-F	ENG-F
24	CHI	Self-Study	BUSS	PHY	CS/ECON	ECON	BIO/ECON/Extra-ENG	ECON/Extra-ENG/ACC
25	ENG	ENG	ENG	ENG	MATH	MATH	PHY	BUSS
26	PHY	CHEM	BUSS	BIO/ECON/CS	CS/ECON	ECON	BIO/ECON/Extra-ENG	ECON/Extra-ENG/ACC
27	ECON/CS/BIO	ECON/BIO/CS	Self-Study	MATH	PHY/BIO	BIO/PHY	PHY	Self-Study
28	COUN	CHI	ACC	BIO/ECON/CS	MATH	MATH	PSY	ENG-F
29	ECON/CS/BIO	ECON/BIO/CS	COUN	PHY	CHEM	CHEM	CHEM	BUSS
30	MATH	PHY	Self-Study	PHY	CHEM	CHEM	CHEM	MATH
31	PSY	CHEM	ACC	MATH	PHY/BIO	BIO/PHY	MATH	Self-Study
32	CHEM	MATH	Self-Study	Self-Study	CS/ECON	ECON	BIO/ECON	ECON/ACC
33	PHY	BIO	MATH	MATH	COUN	ENG-F	MATH	Self-Study
34	Self-Study	PSY	CHI	MATH	ENG-C	ENG-F	PHY	Self-Study
35	ENG	ENG	ENG	ENG	MATH	MATH	ENG-F	MATH
36			ACC	CHEM	CHEM	CHEM	CHEM	MATH
37	MATH	PHY	Self-Study	PSY	CHEM	CHEM	CHEM	Self-Study
38	MATH	PHY	Self-Study	COUN	ENG-C	PSY	MATH	BUSS
39	ECON/CS/BIO	ECON/BIO/CS			MATH	MATH	PHY	
40					ENG-F	ENG-C		

Figure 11. Overall schedule

The overall schedule is structured with each colour demonstrating a cross-module course as shown in Figure 11. Although it is visible that several courses with the same name and different colours, indicating that it is not belong to a cross-module course, occur in the same period, the validity of the schedule is still guaranteed, as these courses are taught by different teachers in different classrooms that do not conflict.

Table 2. Schedule of a single module E

Module E Schedule					
	Mon	Tue	Wed	Thu	Fri
1	PHY/BIO	MATH	MATH	MATH	COUN
2	MATH	PHY/BIO	CS/ECON	CS/ECON	ENG-C
3	MATH	PHY/BIO	PHY/BIO	PHY/BIO	MATH
4	ENG-F	PSY	ENG-F	MATH	CHEM
5	PHE	CS/ECON	CHEM	CHEM	CHEM
6	PHE	CHEM	CHEM	CHEM	ENG-C
7	PHY/BIO	CS/ECON	MATH	PHY/BIO	MATH
8	CHEM	CS/ECON	CS/ECON	CS/ECON	ENG-F

Table 2 shows the schedule of a single module. The parallel courses are separated with a slash.

Experiments are taken based on the course configuration of the past six schedules that are of the same requirement as the current data; the latest data is selected for the illustration. To best reveal the time efficiency of the system to further confirm its feasibility, the program is repeatedly operated 1000 times, and the time consumption of each run is recorded. The central tendency of the system is revealed by the mean and median values, which are 0.7781s and 0.7249s, respectively. The dispersion degree, on the other hand, is exposed by the variance of 0.05158 s^2 , and correspondingly, the standard deviation of 0.2271s, demonstrating the consistency of the systems temporal consumption. Meanwhile, it is found that the minimum value is 0.4059s, and the maximum value is 1.9755s.

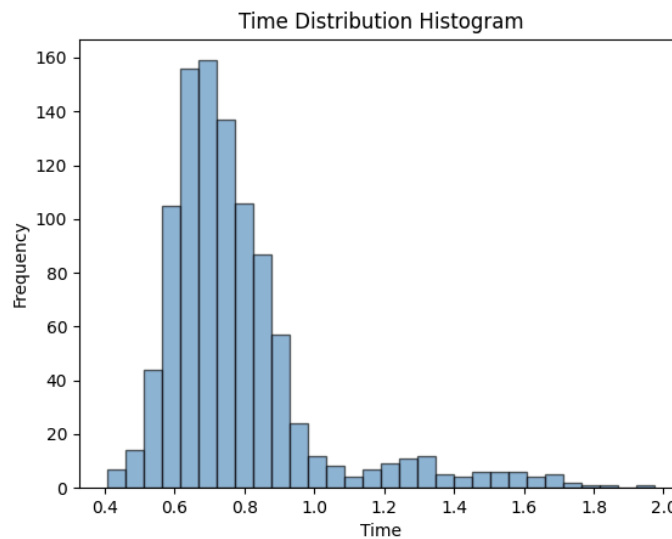


Figure 12. Distribution histogram of system operation time consumption

Figure 12 illustrates the frequency distribution of time consumption, with the x-axis representing time consumed, and the y-axis stands for the frequency. The data is mainly concentrated between 0.6-0.8s, identical with the mean and median value. The figure can be almost considered as a positive skewed normal distribution, besides the minority of data spread nearly anomalously over 1.2-1.8s, exposing the existence of extreme situations.

The concentrated section represents the typical operation time of the algorithm in most situations, showing the general stability of the system. The minor anomaly, however, is considered a consequence of backtracking. The progressive generation, per se, is the process of occupying and diminishing the feasible solution set space, which brings a probability that the praevia solution has already eliminated the possible solution of the current stage, resulting in the inevitable status rollback and the ultimate increment in time consumption.

The proposed system achieves substantially higher efficiency compared to traditional manual course scheduling, which, on average, takes 2.5 days, with an effective time saving of 99%. This suggests that the automatic scheduling system has satisfied the requirement of reducing working hours while maintaining or even surpassing the original schedule's feasibility.

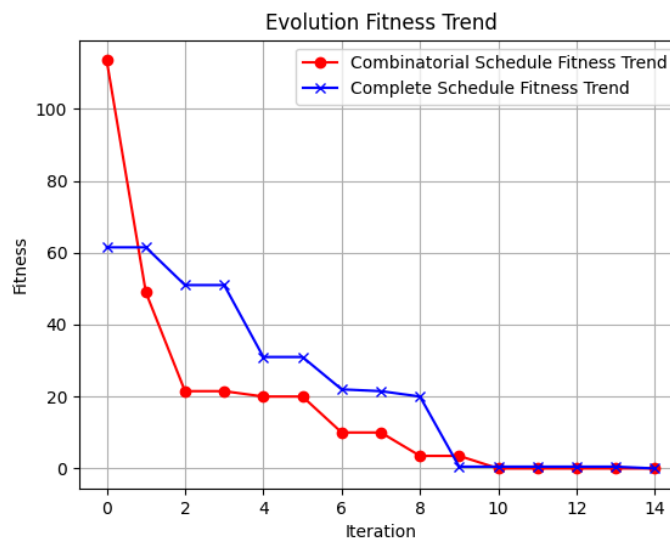


Figure 13. Evolution fitness trend

The fitness trend across revolutions of both combinatorial schedules and complete schedules is recorded as shown in Figure 13 to evaluate the convergence performance of the algorithm. The fitness value of the best individual or each iteration among the population is recorded from the initial population till the feasible solution is obtained.

The maximum iteration time is 14, labeled in x-axis; the fitness is recorded on the y-axis, with lower fitness indicating better feasibility. The red line represents the best fitness of combinatorial schedules, and the blue line represents the best fitness of the complete schedules. In general, a gradual decrement of fitness on both schedules is shown, with a consequent steady tendency and a penalty-free schedule ultimately.

From the initial group to the 4th iteration, the fitness values of both schedules witness a plummet, from around 170 and 60 to 30 and 20, respectively, as a result of the high coefficient hard constraints being rapidly resolved. During the interim progress of the evolution from the 5th to the 9th iteration, the degree of decrease of fitness in both schedules is decreased and approaches a relatively stable

status. At the final stage between 10th -14th iterations, both curves become steady with merely soft constraints existing, resulting in the final feasible solution.

It is revealed that, with the simplification of the problem, the combinatorial schedules declined vigorously at the earlier stage, with some periods stuck temporarily due to the lack of convergence capability during the interim and later stages. As a consequence of the shrinkage of solution spaces in the earlier development. Meanwhile, the complete schedule evolution demonstrates stage-wise stuck, with each of a small period. This can be attributed to the diminishing of solution spaces during the combinatorial iterations.

4. Conclusion

In summary, the system provided a feasible solution to address scheduling difficulties in complex educational scenarios with combinatorial courses of varying length and an indefinite number of parallel courses coexisting. In the system, the Hierarchical Progressive Genetic Algorithm (HPGA) is designed to perform separate operations on different hierarchical genes. Meanwhile, the traditional encoding is modified to be a compatible sequential encoding with the inter-course relationships of the chromosome. Rank-based Roulette Selection and Steady-State Model are incorporated to enable stable population iteration and prevent premature convergence.

On realistic education scenario data, the system completes execution for an average of 0.7781s, providing penalty-free schedules with both hard and soft constraints resolved, with 100% success rate in the current circumstance. The robustness of the system is proved when schedules of more than one grade are progressively and successfully generated in each execution, with the success rate guaranteed by the backtracking mechanism.

However, several limitations of the current system are exposed, which enquire future advancement. Firstly, the main deficiency of the system is that it is only validated when the total course count of combinatorial courses does not exceed the total number of course periods, which can be a potential risk when altering data. Meanwhile, the system requires manual input of the combinatorial courses arrangement, which could be improved to further construct a fully automated system.

References

- [1] Even, S., Itai, A., & Shamir, A. (1975). 16th annual symposium on foundations of computer science (sfcs 1975). *16th Annual Symposium on Foundations of Computer Science (SFCS 1975)*, 184–193.
- [2] Cook, S. A. (1971). The complexity of theorem-proving procedures. *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, 151–158.
- [3] Impagliazzo, R., Paturi, R., & Zane, F. (1998). Which problems have strongly exponential complexity? *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*, 653–663.
- [4] De Coster, A., et al. (2022). Algorithm selection and instance space analysis for curriculum-based course timetabling. *Journal of Scheduling*, 653–672.
- [5] Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 268–308.
- [6] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 671–680.
- [7] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 533–549.
- [8] Dokeroglu, T., & Kucukyilmaz, T. (2024). *A survey on pioneering metaheuristic algorithms between 2019 and 2024*. arXiv:2501.14769
- [9] Aljubarah, A. (2025). From heuristics to hyperheuristics: Comparative study and real-world impact in optimization. *Methodology and Applications in Analysis*

- [10] Jackson, W., Özcan, E., & John, R. I. (2017). Tuning a simulated annealing metaheuristic for cross-domain search. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1055–1062).
- [11] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.
- [12] Rudolph, G. (1994). Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 96–101.
- [13] Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5), 493–530.
- [14] Michalewicz, Z., & Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1), 1–32.
- [15] Homaifar, A., Lai, S. H. Y., & Qi, X. (1994). Constrained optimization via genetic algorithms. *SIMULATION*, 62(4), 242–254.
- [16] Tsang, E., & Jin, N. (2006). Incentive method to handle constraints in evolutionary algorithms. In *Proceedings of the 9th European Conference on Genetic Programming (EuroGP 2006)* (pp. 133–144).
- [17] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press.
- [18] Whitley, D., & Kauth, J. (1988). GENITOR: A different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence* (pp. 118–130).
- [19] Sankaran, P., & McConky, K. (2024). Rethinking selection in generational genetic algorithms to solve combinatorial optimization problems: An upper bound-based parent selection strategy. *arXiv:2410.03863*.
- [20] Syswerda, G. (1991). A study of reproduction in generational and steady state genetic algorithms. In G. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 94–101). Morgan Kaufmann.
- [21] Yilmaz, M. E. (2021). *Analysis of selection methods used in genetic algorithms*.
- [22] Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. *Proceedings of the 1st International Conference on Genetic Algorithms* (pp. 101–111). Lawrence Erlbaum Associates.