LVI SBPO SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
DE 04 ATÉ 07 NOVEMBRO 2024
FORTALEZA | CEARÁ

Comparative Study of Genetic Algorithms for the School Timetabling Problem

Rodrigo Filippo Dias
Rafael de Magalhães Dias Frinhani
Universidade Federal de Itajubá - UNIFEI
Av. BPS, 1303, Pinheirinho. Itajubá, MG
rodrigofilippo@gmail.com; frinhani@unifei.edu.br

Carlos Eduardo de Andrade
AT&T Labs Research
Middletown, NJ, United States
cea@research.att.com

ABSTRACT

The School Timetabling Problem (STP) aims to allocate subjects to specific times for viable planning while avoiding scheduling conflicts between professors, classes, and other resources. We observed this problem in a higher education institution and, to solve it, implemented the genetic algorithm (GA) and variants RKGA (Random-Key Genetic Algorithm) and BRKGA (Biased Random-Key Genetic Algorithm). Different parameterizations were experimented with using real data. We compare the developed methods considering criteria such as the quality of the solutions and execution time. The results show that BRKGA obtained feasible schedules and better quality solutions than RKGA in 76.67% of the cases and all cases compared to GA.

KEYWORDS. ***School Scheduling Problem*, *Genetic Algorithms*, BRKGA.**

Topics (13. MH – Metaheuristics, 14. OC – Combinatorial Optimization)

1. Introduction

Scheduling subjects in educational institutions presents a complex challenge due to the various classes, professors, and constraints involved. Creating timetables is typically done manually, demanding significant effort and expertise from academic coordinators. This manual approach often leads to suboptimal solutions, including overlapping subject schedules, and challenges in allocating professors according to their preferences.

In this scenario, methods like Genetic Algorithms and their variations show great promise for optimizing academic scheduling. These methods can consider constraints and produce feasible and practical solutions that align with student's educational needs and professor's preferences. Automating this process reduces manual labor, enhances the quality of the generated timetables, and establishes a more structured academic environment tailored to educational requirements.

Many federal higher education institutions still rely on manual methods for scheduling subjects. This process is time-consuming and places a significant burden on the managers involved in creating a workable schedule. Managing many classes and professors while accommodating various restrictions and preferences becomes more challenging with a manual approach. Implementing an automated scheduling system for courses helps address organizational constraints related to resource management and compliance, pedagogical concerns impacting student performance, and personal factors like scheduling conflicts with other professor activities (e.g., administrative, research, extension).

This paper proposes a solution for scheduling subjects for two courses at a higher education institution. Additionally, it aims to conduct a comparative study of evolutionary metaheuristics based on the Genetic Algorithm to address the School Timetabling Problem. We are operating on the hypothesis that utilizing variants of the Genetic Algorithm can enhance solution quality and performance compared to the original method. To the best of our knowledge, based on a recent search and the paper of Tan et al. [2021], we do not find works that have used the RKGA or BRKGA variants in solving the School Timetabling Problem (STP).

This paper is structured as follows: Section 2 provides the theoretical foundation, encompassing key concepts, related research, and approaches employed in STP, focusing on genetic algorithms and their adapted versions. Section 3 formally outlines the definition of STP, while Section 4 elaborates on the adopted methods and developed algorithms. Section 5 showcases the results from the experiments, and finally, Section 6 presents the conclusions and outlines future work.

2. Theoretical Reference

Scheduling issues are prevalent in public and private institutions across various sectors, including education, transportation, healthcare, industry, sports, and others [Bettinelli et al., 2015]. These issues involve the task of efficiently assigning a set of events (such as subjects, itineraries, games, surgeries) and resources (such as professors, drivers, athletes, doctors) to specific physical spaces (such as rooms, courts, offices) and timeslots [Zhang et al., 2010]. Despite the common challenge of allocating conflicting items, scheduling problems are further categorized based on the specific context.

Particularly in the academic scenario, the scheduling problem pertains to creating timetables for educational institutions like schools, colleges, and universities [Teoh et al., 2015]. Each new academic semester requires course managers to establish class schedules, balancing pedagogical needs, organizational constraints, and professor's preferences. Institutions' unique operational dynamics and educational levels have given rise to various academic scheduling problems. According to Willemen [2002], these problems can be further classified into subcategories such as Course Timetabling, Examination Timetabling, University Timetabling, and School Timetabling.

The STP involves assigning each subject to specific times while avoiding scheduling conflicts. These conflicts may include scheduling the same professor for multiple classes simultaneously or scheduling different subjects for the same class simultaneously, among other constraints [Tan et al., 2021]. STP is a challenging combinatorial optimization problem classified under the NP-Hard category [Pillay, 2014], indicating that no deterministic polynomial time algorithms are currently available for its solution. Due to its complexity, manually obtaining a feasible solution can be time-consuming or impractical, mainly when dealing with numerous elements such as professors, classes, and constraints.

Various methods solve the STP by using strategies like Integer Programming [Fonseca et al., 2017], single solution metaheuristics such as Simulated Annealing [Zhang et al., 2010] and Tabu Search [Minh et al., 2010] and population metaheuristics such as Particle Swarm [Tassopoulos and Beligiannis, 2012] and Genetic Algorithm [Raghavjee and Pillay, 2015].

2.1. Genetic Algorithm and their variants

In the context of bioinspired algorithms, the Genetic Algorithm (GA) is one of the best known. It emerged in the 1960's to solve complex problems by simulating evolutionary concepts such as natural selection of the fittest individuals and mutations [Holland, 1992]. The typical procedure for traditional GA's involves generating an initial population of solutions randomly, statically, or through a combination of both. Genetic crossover and mutation operators are then applied to this population to create the next generation of solutions. Like genetics, crossover always occurs, while mutation only happens with a small probability. The chromosome of an individual corresponds to a solution to the problem.

For more clarity, Figure 1(a) illustrates a population of individuals x with identifiers ranging from 0 to n. A binary vector commonly represents the individual's chromosome. Each element in the vector corresponds to a gene that expresses a characteristic of the individual, with its value indicating the allele. An individual has a fitness value associated with him, given by the model's objective function (e.g., $f(x_0)$ for the solution x_0).

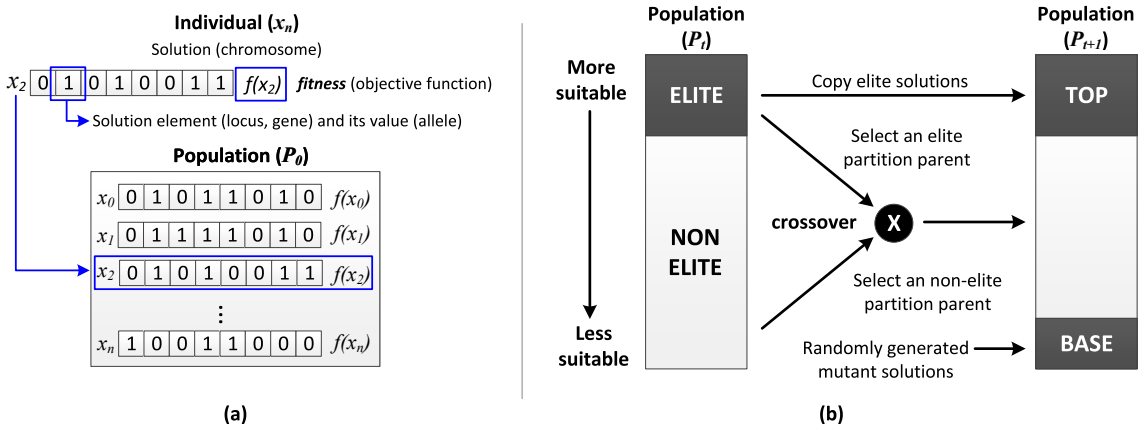

Figure 1: In (a) representation of an individual and a population. In (b) structure of population in RKGA and BRKGA variants. Font: adapted from Gonçalves and Resende [2011].

In Figure 1(b), we can see the composition of a population, representing a group of individuals. The individuals reproduce using crossing mechanisms to generate a new population in each generation. The latest population's individuals are sorted based on their fitness function, with the fittest individual at the top and the worst fit at the bottom. The elite set is always carried over to the next generation and comprises the top $w\%$ fittest individuals. Although the population structure of

RKGA and BRKGA is similar, the variants differ in the parents' choice. The RKGA can generate a new individual by combining two non-elite individuals. In contrast, in BRKGA, at least one of the individuals must be from the elite group. Mutants randomly generated are introduced into the population to enhance diversity and prevent the algorithm from stagnating in local sub-optima.

The crossover or recombination process is a fundamental step in genetic algorithms that combines genetic information from two or more individuals to generate offspring. Crossover aims to explore different combinations of genetic characteristics present in individuals in the population and create new, potentially better solutions. Figure 2(a) illustrates the *2-point* crossover, in which two cutoff points are randomly selected in the genetic representation of the parents, followed by the division of the sequence into three segments: initial, central, and final. In a *Multi-point* crossover, Figure 2(b), multiple points are randomly determined to segment the arrays of the parents. For each segment, a parent is randomly chosen to contribute to the descendant constitution.

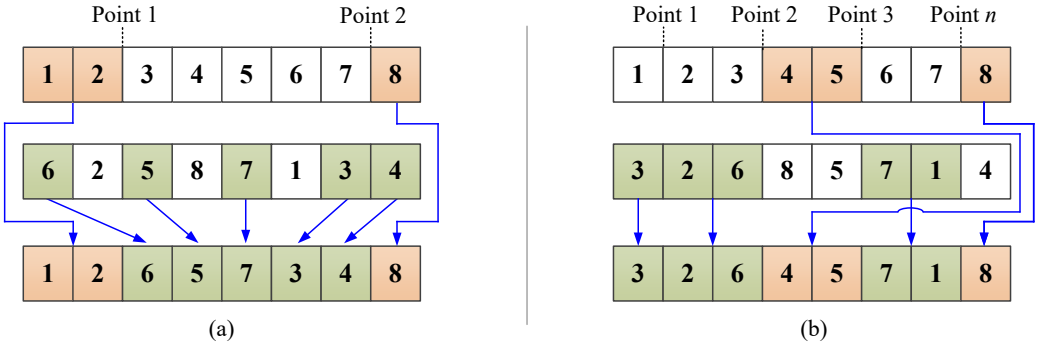

Figure 2: Visual representation of the crossover cutting points in (a) *2-point* and (b) *Multi-point* to transfer of parents' characteristics to the child.

Various adaptations of the traditional method of Genetic Algorithms have emerged, including the Random-Key Genetic Algorithm (RKGA) [Bean, 1994] and the Biased Random-Key Genetic Algorithm (BRKGA) [Gonçalves and Resende, 2011; Frinhani et al., 2015]. One key distinction between RKGA and BRKGA concerning GAs algorithms is their dividing individuals into elite and non-elite sets. This approach contributes to improved optimization problem outcomes. Figure 3 illustrates the execution flow of BRKGA, which uses coding based on random keys to represent individuals in the population.

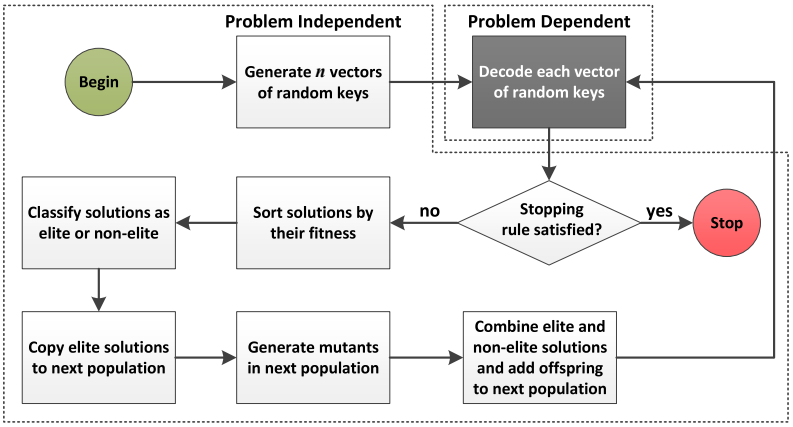

Figure 3: BRKGA algorithm execution flowchart. Font: adapted from Gonçalves and Resende [2011].

Initially, the algorithm is configured with parameters such as population size, crossover rate, and mutation rate. The initial individuals are generated as random keys, which are sequences of positive real numbers in the range [0, 1], then decoded. Currently, the BRKGA-MP-IPR [Andrade et al., 2021] is the state-of-the-art of GA's variants. Through a deterministic function, it is possible to calculate the fitness of each individual of the population and measure how fit it is for solving the problem. If the stopping criterion has not been met (e.g., number of generations), the individuals are ordered according to their fitness and classified as elite or non-elite. The subsequent population receives the elite individuals and those generated by crossover and mutation.

3. Problem Definition

The study takes a practical and quantitative approach to the concept of STP, and it is an exploratory case study. It was conducted at the Institute of Mathematics and Computing (IMC) at the Federal University of Itajubá (UNIFEI) for the Computer Science (CCO) and Information Systems (SIN) courses. The IMC has around 18 computing professors and 450 active students across both courses. The subjects are organized into periods based on the pedagogical projects of the courses. CCO offers 41 subjects distributed over eight periods, and SIN offers 44 subjects distributed over eight periods. Both courses provide odd-numbered period subjects in the 1^{st} semester and even-numbered period subjects in the 2^{nd} semester.

A matrix organizes the time slots and shifts in which each subject is offered, depending on the availability of professors. Each course has a matrix for each period provided in the semester. The CCO course operates on an integral schedule, with classes allocated to the morning, afternoon, or evening shifts. The morning and afternoon shifts receive priority. The SIN course occurs on the night shift only. Each shift has 5 slots of time totaling 15 slots of time per day. In day shifts, each slot corresponds to a class lasting 55 minutes, and the night shift corresponds to one class lasting 50 minutes. The subject classes are scheduled from Monday to Saturday, prioritizing until Friday whenever possible. Courses with 2 credits require 2 consecutive slots per week; those with 4 credits require 4 slots/week (2 successive slots in two days or 4 consecutive slots in 1 day).

The following non-linear mathematical model formalizes the problem for the study scenario. The general idea is to minimize the weighted sum of violations of the problem's constraints via a set of weights that measure the intensity of each breach according to the user's needs. This approach has been chosen due to the difficulty of finding viable solutions when all constraints are imposed. When transferring these constraints to the objective function, the problem is relaxed, and every solution is feasible from the model's point of view, facilitating the construction of methods for its resolution. Additionally, practical solutions that, although unfeasible, are still good enough for university planners to appreciate may be obtained.

Let T be the set of classes or subjects, P be the set of professors, and $H = \{1, \ldots, n\}$ be the set of schedules. Consider $T^2 \subseteq T$ and $T^4 \subseteq T$, such that $T^2 \cap T^4 = \emptyset$ as the sets of subjects of two and four credits respectively. Consider the credit vector $K_t \in \mathbb{N}$ such that $K_t = k$ for $t \in T^k$. The value of each cost constraint described below equals the product of the constraint's weight by the total number of times that's violated in the solution. The value of $C_{ph}^{PrefTimes} \in \mathbb{N}$ indicates the cost of violating the time preferences $h \in H$ of each professor $p \in P$. $C_{ph}^{ProhibTimes} \in \mathbb{N}$ is the cost resulting from assigning times $h \in H$ considered prohibited or not suitable for a given professor $p \in P$. The value $C^{AvoidClashes}$ corresponds to the cost of violating the constraint in which a professor p is simultaneously assigned to different classes t in the same time slot. The value $C_{ht}^{FixedTime} \in \mathbb{N}$ corresponds to the cost of violating fixed time schedules and aims to ensure that a course subject does not occur in the wrong shift. Therefore, this weight is zero for the times $h \in H$ allowed for class $t \in T$. In other cases, the weight is adjusted to a positive value. The value of $C^{AvoidSplit} \in \mathbb{N}$ indicates the cost of allocations that do not have at least two consecutive slots

of schedules. The binary decision variable x_{pth} indicates whether professor $p \in P$ is assigned to class $t \in T$ at time $h \in H$. The binary decision variable y_{th} indicates whether the class $t \in T$ is assigned to the time slot $h \in H$.

The Objective Function (1) seeks the minimum value solution considering the weighting of the violation of constraints given their costs. The first term considers each professor's preferred schedules, being zero cost if all of their preferences have been met. The smaller $C^{PrefTimes}$ is, the more suitable that timetable is. The second term penalizes assignments for times when professors are not available. Note that the first and second terms can be carefully combined by adjusting their weights. The third term penalizes collisions, which occur when the same professor is assigned to different classes in the same slot time. The fourth term of the objective function punishes assignments that violate previously fixed timetables, with the cost being a positive value when the time h should not be assigned to time t. Finally, the fifth and sixth terms penalize solutions that assign two or four credit courses at non-consecutive times. For two-credit courses, check pairs of non-contiguous *slots*. The verification is similar in cases with four credits but divided into two blocks or two pairs of *slots*. Note that although both terms are non-linear, they can be linearized by introducing auxiliary variables, but for brevity, this is not covered in this text.

$$
\begin{aligned}
Minimize \quad & \sum_{p\in P}\sum_{t\in T}\sum_{h\in H} C_{ph}^{PrefTimes}\cdot x_{pth} \quad + \\
& \sum_{p\in P}\sum_{t\in T}\sum_{h\in H} C_{ph}^{ProhibTimes}\cdot x_{pth} \quad + \\
& \sum_{\substack{t_1,t_2\in T:\\ t_1\neq t_2}}\sum_{p\in P}\sum_{h\in H} C^{AvoidClashes}\cdot (x_{pt_1h}\cdot x_{pt_2h}) \quad + \\
& \sum_{h\in H}\sum_{t\in T} C_{ht}^{FixedTime}\cdot y_{th} \quad + \\
& \sum_{t\in T^2}\sum_{\substack{h,h'\in H:\\ h<h'+1}} C^{AvoidSplit}\cdot (y_{th}\cdot y_{th'}) \quad + \\
& \sum_{t\in T^4}\sum_{\substack{h^1,\ldots,^4\in H:\\ h^1<\ldots<h^4,\\ h^1<h^2+1,\\ h^3<h^4+1}} C^{AvoidSplit}\cdot (y_{th^1}\cdot y_{th^2}+y_{th^3}\cdot y_{th^4}) \quad +
\end{aligned}
\tag{1}
$$

Subject to:

$$\sum_{h\in H} y_{th} \geq K^k, \quad \forall\, t\in T^k, k\in\{2,4\}, \tag{2}$$

$$\sum_{p\in P} x_{pth} = y_{ht}, \quad \forall\, t\in T, h\in H. \tag{3}$$

As many practical constraints were relaxed due to your inclusion in the objective function, only two specific constraints are considered. The Constraint (2) ensures the number of time slots allocated to a subject conforms to its number of credits (two or four). Constraint (3) guarantees that

if a class is assigned to a specific time, a professor will also be assigned to that time. It is important to note that this model allows planners great flexibility in adjusting their preferred constraints. However, care and ingenuity are necessary when determining weights so that preferences can reflect the desired reality. From a resolution point of view, it is possible to develop new methods to minimize the violation of practical constraints modeled in the objective function, enabling the use of a broader range of techniques.

4. Implemented Methods

The methods implemented were the traditional Genetic Algorithm (GA) and the RKGA and BRKGA variants, with the crossing strategies *2-point* and *Multi-point*. A key-value structure and matrices represent the individual. The key uniquely identifies the individual, and the value corresponds to the timetable matrices of the semester periods under analysis, in addition to the corresponding fitness function. As an example, an individual with key (*id*) equal to 0 is made up of four matrices, one for each period of the course (e.g., in the 1^{st} semester of the year, the 1^{st}, 3^{rd}, 5^{th}, and 7^{th} periods). Another key-value structure stores the individuals of a population (generation), being a mutable, indexed, and unordered collection of elements.

The initial population (P_0) corresponds to generation 0, being constructed entirely randomly until reaching a parameterizable criterion (e.g., maximum number of individuals). The next generations are created through crossing, mutation, and selection of individuals. A key-value structure stores the generated populations, with the key identifying a specific generation and the value corresponding to the population. It is possible to locate the individual and obtain their characteristics from the chain of individual, population, and generation structures. This format contributes to executing population evolution operations throughout the algorithm iterations.

For GA, two approaches were used. The first follows the original strategy, randomly selecting the parents considering the entire population, followed by their crossing. The second procedure is analogous to BRKGA, where one parent is necessarily chosen from the elite set, and the other is randomly selected from the entire population. The individuals in the new generation are ordered according to the fitness function, with the best ranked $x\%$ being considered elite and copied to the next generation. The solution obtained at the end of the algorithm execution is the one that violates the least *hard* and *soft* restrictions. RKGA and BRKGA also use this strategy in their implementations, but the decoder and random keys are used only in the variants. Considering matrix modeling, for coding to occur, each matrix is first converted into a one-dimensional vector. After conversion, the vectors are encoded, and the random keys are sorted in ascending order. A copy of the subject vectors and respective codes is made to enable subsequent decoding of the solution.

The crossover and mutation methods were adjusted considering the use of matrices to represent timetables. Crossover starts with converting the selected matrix from each parent into a vector. Then, randomly are choosed cutting points in the vector of the parents. A daughter vector is created from these points by combining the genes of the first parent, which are at the extremes of the cutoff points, and those of the second parent, which are between the cutoff points. Furthermore, a conditional check is carried out to ensure that subjects are classified, respecting their maximum credit specifications. Although there are multiple occurrences of the same subject in the timetable matrix, there is a credit limit that must not be exceeded. This restriction is taken during the crossover process to avoid excessive workload allocation and the creation of feasible solutions.

About the *2-point* crossover and *Multi-point* crossover methods, their main distinction is in the way the parent matrices are selected. In *2-point*, the matrices are chosen randomly, while in *Multi-point*, all matrices are subjected to the crossover process, allowing a broader exploration of possible genetic combinations. The Algorithm 1 corresponds to *2-point* crossover, which has as input the matrices corresponding to the *parent1* and *parent2* chosen randomly. In line 1, a vector is

allocated for the child individual, whose dimension is the same as that of the parents. The matrix of each parent is converted into a vector (lines 2 and 3), and the crossover cutoff points are defined randomly (lines 4 and 5). The purpose of the ties is to attribute to the child the characteristics of the first parent (*parent1*), given by the genes between position 0 and the position of *point_one* (lines 6–7), as well as those from the position of *point_two* to the final position of the vector (lines 8–9). Similarly, the same procedure mentioned above is carried out for the second parent through the loop of lines 10–12, which attributes to the child its characteristics, given by the genes included between the positions *point_one* and *point_two*. To prevent duplicates when adding parent characteristics to the child and ensure feasible solutions, line 11 only adds the feature if it is not already present.

Algorithm 1: *2-point crossover*

```
   Data: parent1, parent2
 1 child ← [];
 2 parent1_array ← toArray(parent1);
 3 parent2_array ← toArray(parent2);
 4 point_one ← random(0, length(parent1_array)/2);
 5 point_two ← random(length(parent1_array)/2, length(parent1_array));
 6 for i in range(0, point_one) do
 7 |   child[i] = parent1_array[i];
 8 for j in range(point_two, lenght(parent1_array)) do
 9 |   child[j] = parent1_array[j];
10 for k in range(point_one +1, point_two −1) do
11 |   if parent2_array[k] ∉ child then
12 |   |   child[k] = parent2_array[k];
13 return child;
```

In the *Multi-point* crossover, all matrices from the first parent will undergo crossover with the matrices from the second parent. This way, more significant variability is guaranteed, being useful mainly for large datasets. In an individual with several matrices, if the crossover is carried out in just one of the matrices, the evolution of the Population will take longer, requiring more generations due to the slight difference between individuals and a longer time until convergence.

In mutation, the random permutation strategy is adopted, in which each individual has a parameterizable mutation rate of $y\%$. When an individual is selected to undergo the mutation, a random permutation is applied to one of your matrices. In this way, a new individual is generated without using parents, as occurs in crossover. It is essential to highlight that at least one individual in the population is subjected to the mutation process, ensuring diversification and exploring different genetic configurations. The mutation rate determines the probability of a gene being changed during the mutation process, promoting the introduction of variations into the population and avoiding stagnation at local optima.

5. Experiments

The language *Python* version 3.8.5 was used to develop the methods. The machine used for the experiments contained an Intel(R) Core(TM) i7-9750H processor, 16.0 GB of RAM, 1TB of HD memory, Windows 10 64-bit operating system. We carried out the experiments in the same development environment.

Eight methods were evaluated: GA with *2-point crossover* and without elitism (GA_{2p}), GA with *2-point* and elitism (GA_{2pE}), GA with *Multi-point crossover* without elitism (GA_{Mp}) and with elitism (GA_{MpE}), RKGA with *2-point* ($RKGA_{2p}$) and BRKGA with *2-point* ($BRKGA_{2p}$), RKGA with *Multi-point* ($RKGA_{Mp}$) and BRKGA with *Multi-point* ($BRKGA_{Mp}$) . The param-

eters defined for the execution of the methods were mutation rate (0.05%), number of individuals (10, 15, 20, 50, 100, and 200), and number of generations (10, 50, 100, 200, and 400).

The metrics used to evaluate the solutions were: (*i*) Fitness (evaluates how appropriate the solution is according to the established criteria); (*ii*) a Total number of restrictions violated (the lower, the better the quality); (*iii*) Number of violated *soft* constraints (soft constraints related to professor's preferences that can be violated but still affect the quality of the solution); (*iv*) Number of *hard* restrictions violated (mandatory restrictions considered essential that were violated, whose high value indicates a lower quality solution); (*v*) Execution time of the method measured in seconds until presenting the final solution. A solution considered adequate must be found in a reasonable time (less than an hour) and minimize the number of *soft* and *hard* constraints violated to obtain a feasible and quality matrix of schedules.

5.1. Results

Table 1 contains the results of the experiments in which the column # Indiv. shows the number of individuals in the population, the columns *HR* and *SR* respectively contain the number of *Hard* and *Soft* constraints that were violated, *F0* the score of the solution found and t the time necessary until the completion of the method. The results indicate that as the population and the number of generations increases, there is a progressive improvement in the quality of the solutions found. This behavior suggests that exploring a broader search space provided by a larger population and with more opportunities for evolution can contribute to discovering better solutions.

Analysis of the schedules generated by $BRKGA_{Mp}$ revealed a significant conflict reduction. There is a better use of resources, with a more balanced distribution of subjects in the timetable slots and reduced overlaps between professors and classes. A gradual refinement of solutions over the generations was also evident, indicating the evolution of the population in reducing remaining conflicts and optimizing allocations. In $BRKGA_{Mp}$, gradual improvements in the solutions can be seen, with fewer constraints being violated over the generations. For example, the case with 20 individuals and 200 generations that obtained a solution without any *hard* constraint violated, and 100 generations, which obtains only 1 *HR* and 4 *SR*.

In some cases (e.g., GA_{2p} and $RKGA_{2p}$ with 200 individuals), no significant improvements were observed, even with the increase in populations and generations. Another finding revealed that using elite individuals in GA did not provide substantial benefits. The solutions obtained had similar costs in many cases, but the time needed to find them varied. We can observe this pattern in both crossing strategies adopted. An example would be GA_{2p} and GA_{2pE} with 100 generations, whose solutions presented the same objective function value and violated the same number of constraints, but the execution time increased. In contrast, BRKGA demonstrated superior performance compared to GA and RKGA, finding solutions without violating *hard*, achieving satisfactory results in minutes. The elite parent selection strategy did not present significant benefits for GA_{2p} and GA_{Mp}, while BRKGA obtained better results than RKGA in the two crossover methods analyzed. We noted that although RKGA was unable to find feasible solutions, $RKGA_{Mp}$ with 50 generations and 100 individuals obtained solutions with few *soft* and *hard* constraints violated. This situation highlights the relevance of not immediately discarding unfeasible solutions, as they can still contribute to improvements.

The *Multi-point* strategy proved more efficient and robust than the *2-point* crossover. Performing crossover on all matrices one at a time resulted in superior solutions compared to the purely *2-point*-based strategy. This behavior indicates that considering a more significant number of crossover points when combining chromosomes can lead to more promising and diverse solutions. Another advantage of *Multi-point* is its ability to generate more significant variability due to increased crossing points between parents.

Table 1: Experiment results of the implemented methods.

		GENERATIONS																	
		10				50				100				200				400			
METHOD	# Indiv.	*HR*	*SR*	*F0*	*t*	*HR*	*SR*	*F0*	*t*	*HR*	*SR*	F0	*t*	HR	*SR*	*F0*	*t*	*HR*	*SR*	*F0*	*t*
GA 2 point no elite	10	5	20	5020	1,57	6	24	6024	1,57	4	22	4022	1,36	4	24	4024	2,73	3	18	3018	5,47
	15	8	38	8038	2,30	7	22	7022	1,08	7	20	7020	2,13	4	12	4012	4,33	4	22	4022	8,58
	20	6	26	6026	3,08	5	15	5015	1,34	5	15	5015	2,68	4	12	4012	5,23	2	22	2022	10,46
	50	6	27	6027	7,14	6	24	6024	3,36	6	24	6024	6,81	4	12	4012	12,23	4	12	4012	258,00
	100	7	19	7019	1,41	6	22	6022	6,48	6	22	6022	12,85	3	18	3018	259,91	4	12	4012	512,20
	200	7	23	7023	2,83	3	22	3022	8,23	3	22	3022	262,50	3	18	3018	509,57	3	22	3022	1.018,34
GA 2 point with elite	10	5	20	5020	1,52	6	24	6024	3,67	4	22	4022	1,39	5	15	5015	2,76	3	18	3018	5,42
	15	8	38	8038	2,36	7	22	7022	1,22	7	20	7020	2,17	6	24	6024	4,23	4	22	4022	8,54
	20	6	26	6026	2,92	5	15	5015	1,56	5	15	5015	2,65	6	22	6022	5,30	2	22	2022	10,51
	50	6	27	6027	7,12	6	24	6024	3,76	6	24	6024	6,49	3	22	3022	130,34	4	12	4012	258,08
	100	7	19	7019	1,51	6	22	6022	7,44	6	22	6022	12,97	4	24	4024	261,01	4	12	4012	514,86
	200	7	23	7023	2,81	3	22	3022	147,39	3	22	3022	264,22	4	12	4012	514,09	3	22	3022	1.032,53
RKGA 2 point	10	3	7	3007	1,63	2	12	2012	1,14	3	7	3007	1,41	3	7	3007	2,83	3	8	3008	5,54
	15	3	7	3007	2,59	3	7	3007	1,15	3	7	3007	2,31	3	7	3007	4,62	2	10	2010	8,90
	20	3	10	2010	3,26	2	10	2010	1,43	2	10	2010	2,79	2	10	2010	5,53	3	6	2006	11,03
	50	3	12	2012	8,10	3	12	2012	3,54	3	12	2012	6,97	2	12	2012	8,23	2	11	2011	275,02
	100	2	12	2012	1,60	2	12	2012	7,02	2	12	2012	5,23	2	12	2012	278,89	2	6	2006	549,22
	200	2	3	2003	3,21	2	3	2003	8,23	2	3	2003	278,53	2	3	2003	562,31	2	4	2004	1.106,62
BRKGA 2 point	10	3	8	3008	1,59	2	11	2011	1,42	3	8	3008	1,41	3	8	3008	2,76	3	18	3018	5,90
	15	2	10	2010	2,60	2	10	2010	1,16	2	10	2010	2,25	2	10	2010	4,51	4	22	4022	8,92
	20	3	6	2006	3,19	3	6	2006	1,60	2	6	2006	2,80	2	6	2006	5,59	2	22	2022	11,32
	50	2	11	2011	7,96	2	11	2011	3,63	2	11	2011	6,94	2	11	2011	137,47	4	12	4012	280,50
	100	2	6	2006	1,63	2	6	2006	7,07	2	6	2006	2,23	2	6	2006	274,29	4	12	4012	570,38
	200	2	4	2004	3,20	2	4	2004	5,23	2	4	2004	284,15	2	4	2004	546,20	3	22	3022	1.103,40
GA Multi-point no elite	10	7	21	7021	1,48	7	14	7014	2,29	5	11	5011	1,36	4	15	4015	2,66	3	25	3025	5,28
	15	8	16	8016	2,29	6	33	6033	1,08	4	27	4027	2,07	4	17	4017	4,19	2	30	2030	8,27
	20	8	19	8019	2,81	7	33	7033	1,30	5	19	5019	2,60	4	30	4030	5,14	4	17	4017	10,19
	50	7	28	7028	6,98	7	14	7014	3,23	7	14	7014	6,30	5	15	5015	12,54	4	23	4023	258,48
	100	5	18	5018	1,38	3	25	3025	6,37	3	25	3025	12,48	3	25	3025	248,69	4	18	4018	507,96
	200	10	19	10019	2,75	5	19	5019	12,82	5	19	5019	260,60	4	21	4021	512,44	3	25	3025	988,80
GA Multi-point with elite	10	4	13	4013	1,44	5	19	5019	2,26	3	11	3011	1,34	3	11	3011	2,68	3	11	3011	5,24
	15	8	21	8021	2,26	5	19	5019	1,05	5	19	5019	2,06	4	28	4028	4,10	4	28	4028	8,19
	20	7	28	7028	2,83	5	26	5026	1,31	2	20	2020	2,64	2	20	2020	5,23	2	20	2020	10,43
	50	7	30	7030	7,04	5	19	5019	3,22	3	24	3024	6,32	3	21	3021	132,22	3	21	3021	249,69
	100	9	35	9035	1,38	5	19	5019	6,48	4	31	4031	12,50	4	22	4022	249,83	2	27	2027	497,94
	200	7	23	7023	2,75	6	24	6024	12,79	5	12	5012	253,41	4	28	4028	548,95	3	26	3026	1.014,83
RKGA Multi-point	10	2	90	2059	1,35	1	24	1024	2,64	1	24	1024	1,17	1	24	1024	2,30	1	24	1024	4,57
	15	2	53	2053	2,14	2	22	2022	8,19	2	44	2044	1,86	2	40	2040	3,72	2	22	2022	7,39
	20	2	90	2090	2,74	2	22	2022	1,18	2	22	2022	2,30	2	22	2022	4,56	2	22	2022	9,18
	50	2	24	2024	6,65	2	22	2022	2,93	2	22	2022	5,76	1	11	1011	11,59	1	11	1011	231,99
	100	2	59	2024	1,38	1	24	**1024**	5,93	1	22	1022	11,44	1	11	1011	230,17	1	11	1011	460,89
	200	2	1	2001	2,62	1	11	1011	11,68	1	1	1011	229,54	1	1	1011	457,30	1	11	1011	908,09
BRKGA Multi-point	10	1	4	**1004**	1,33	1	69	1069	2,19	1	4	**1004**	1,15	1	44	1044	2,29	0	40	40	4,57
	15	2	44	2044	2,16	2	44	2044	2,19	2	44	2044	1,87	2	43	2043	3,70	0	40	40	7,35
	20	1	50	1050	2,68	1	50	1050	1,18	1	50	1050	2,29	0	40	**40**	4,55	0	38	**38**	9,02
	50	1	69	1069	6,69	1	69	1069	3,07	1	69	1069	5,70	0	40	40	666,51	0	38	38	225,60
	100	1	66	1066	1,31	1	66	1066	5,90	1	66	1066	11,45	0	40	40	229,74	0	38	38	470,88
	200	1	64	1064	2,63	1	64	1064	11,58	1	64	1064	239,11	1	64	1064	453,04	0	38	38	931,80

Table 2 contains an example of the solution for the course schedules for classes in odd-numbered semesters (1^{st}, 3^{rd}, 5^{th}, and 7^{th} Periods) of Computer Science and Information Systems courses, according to the solution obtained by $BRKGA_{Mp}$ with populations of 20 individuals and 400 generations. A total of 38 *Preferred_Times* constraints were violated. An example of breaking this constraint would be on a Tuesday, in which Professor P13 would not like to be allocated but ended up being. Although it is not a breach of constraint, it can negatively affect the timetable quality generated in cases of subjects allocated in a sequence of four slots. An example occurs in the 3^{rd} period, in which the subject SRSC03, taught by Professor P11 on Thursday, was allocated four consecutive times. We observe this situation in other cases, such as the CIC271 subject with Professor P07 in the 7^{th} period.

Some cases do not correspond to constraint violations but would imply improvements from a business perspective. For example, the subject XMAC01, taught by Professor P05, was assigned to the first two periods of the morning, but ideally, it would have been allocated to the second and third periods of this shift. The subject COM230 of the 5^{th} period was allocated in the third and fourth-time slots on Monday, but as between these slots there is a rest interval, the ideal would be for the subject to be allocated in the second and third-time slots (2M23) or the fourth and fifth (2M45).

Table 2: Class schedules matrices for the first semester of Computer Science and Information Systems courses (1^{st}, 3^{rd}, 5^{th} e 7^{th} Periods).

1^{st} Period

Turn	Monday	Tuesday	Wednesday	Thursday	Friday
M1	-	XDES01 - P12	-	-	XMAC01 - P05
M2	MAT050	XDES01 - P12	-	-	XMAC01 - P05
M3	MAT050	CRSC03 - P13	-	-	-
M4	MAT050	CRSC03 - P13	CAHC04 - P16	XMAC01 - P05	-
M5	MAT050	-	CAHC04 - P16	XMAC01 - P05	-
T1	MAT011	-	-	MAT00A	-
T2	MAT011	-	-	MAT00A	-
T3	MAT011	CRSC03 - P13	XDES01 - P12	-	MAT00A
T4	MAT011	CRSC03 - P13	XDES01 - P12	-	MAT00A
T5	-	-	-	-	-
N1	SAHC05 - P10	SAHC05 - P10	IEPG22	XDES01 - P15	SAHC04 - P06
N2	SAHC05 - P10	SAHC05 - P10	IEPG22	XDES01 - P15	SAHC04 - P06
N3	MAT00A	MAT00A	XDES01 - P15	-	IEPG01
N4	MAT00A	MAT00A	XDES01 - P15	-	IEPG01
N5	-	-	-	-	IEPG01

3^{rd} Period

Monday	Tuesday	Wednesday	Thursday	Friday
-	-	-	-	MAT002
CMAC03 - P16	XDES04 - P01	CTCO02 - P18	XDES02 - P09	MAT002
CMAC03 - P16	XDES04 - P01	CTCO02 - P18	XDES02 - P09	MAT002
XMAC02 - P09	XMAC02 - P09	CRSC02 - P13	-	CRSC02 - P13
XMAC02 - P09	XMAC02 - P09	CRSC02 - P13	-	CRSC02 - P13
-	-	-	XDES04 - P01	MAT013
-	-	-	XDES04 - P01	MAT013
-	CTCO02 - P18	-	XDES02 - P09	CMAC03 - P16
-	CTCO02 - P18	-	XDES02 - P09	CMAC03 - P16
-	-	-	-	-
STCO02 - P14	XDES03 - P15	STCO02 - P14	SRSC03 - P11	XDES03 - P15
STCO02 - P14	XDES03 - P15	STCO02 - P14	SRSC03 - P11	XDES03 - P15
SDES05 - P08	-	SDES05 - P08	SRSC03 - P11	ECN001
SDES05 - P08	-	SDES05 - P08	SRSC03 - P11	ECN001
-	-	-	-	ECN001

5^{th} Period

Turn	Monday	Tuesday	Wednesday	Thursday	Friday
M1	-	-	-	-	-
M2	-	-	CIC132 - P05	-	COM211 - P08
M3	COM230 - P12	-	CIC132 - P05	-	COM211 - P08
M4	COM230 - P12	COM211 - P08	COM230 - P12	-	-
M5	-	COM211 - P08	COM230 - P12	-	-
T1	COM221 - P10	COM211 - P08	-	CIC132 - P05	-
T2	COM221 - P10	COM211 - P08	-	CIC132 - P05	-
T3	CIC111 - P14	COM311 - P02	COM240 - P04	COM240 - P04	-
T4	CIC111 - P14	COM311 - P02	COM240 - P04	COM240 - P04	-
T5	CIC111 - P14	COM311 - P02	-	-	-
N1	-	SIN311 - P05	COM231 - P18	COM240 - P04	SIN260 - P07
N2	-	SIN311 - P05	COM231 - P18	COM240 - P04	SIN260 - P07
N3	SIN132 - P05	SIN260 - P07	COM240 - P04	SIN311 - P05	SIN132 - P05
N4	SIN132 - P05	SIN260 - P07	COM240 - P04	SIN311 - P05	SIN132 - P05
N5	-	SIN260 - P07	-	-	-

7^{th} Period

Monday	Tuesday	Wednesday	Thursday	Friday
-	-	-	-	COM222 - P15
COM212 - P01	-	-	-	COM222 - P15
COM212 - P01	-	-	COM242 - P16	COM222 - P15
-	COM213 - P06	COM212 - P01	COM242 - P16	COM222 - P15
-	COM213 - P06	COM212 - P01	-	COM222 - P15
CIC271 - P07	-	COM213 - P06	-	-
CIC271 - P07	-	COM213 - P06	-	-
CIC271 - P07	COM212 - P01	-	COM242 - P16	-
CIC271 - P07	COM212 - P01	-	COM242 - P16	-
-	-	-	-	-
SIN210 - P08	SIN210 - P08	COM213 - P06	SIN413 - P08	SIN413 - P08
SIN210 - P08	SIN210 - P08	COM213 - P06	SIN413 - P08	SIN413 - P08
SIN412 - P10	SIN412 - P10	COM213 - P06	SIN414 - P04	SIN313 - P02
SIN412 - P10	SIN412 - P10	COM213 - P06	SIN414 - P04	SIN313 - P02
-	SIN412 - P10	-	SIN414 - P04	SIN313 - P02

Another item with potential for improvement is the excess free time between classes on certain days or the allocation of many continuous slots. A more balanced distribution of times throughout the week would make the schedule less tiring. The solutions respected the course shifts and allocated subjects to at least two consecutive time slots.

6. Conclusions

STP is a complex problem that involves efficiently allocating subjects at specific times to avoid conflicts in using restricted resources, meeting pedagogical requirements, and accommodating professor's preferences. Among the methods implemented for its solution, BRKGA presented the best performance, with the experiments showing satisfactory and feasible results at an acceptable computational cost.

A suggestion for future work would be adopting a restriction that imposes minimum and maximum limits on free time slots between professor's classes. Another would be using alternative crossover strategies other than *2-point* and *Multi-point* to improve the solutions. Also, using adaptive crossing techniques in which operators are dynamically adjusted during the evolutionary process can provide better solutions. Combining multiple crossover operators can increase genetic diversity and the ability to explore the search space in more detail, resulting in more robust and better-quality solutions.

Another improvement is the development of a front-end interface for the end user's friendly use of scheduling software. This interface would facilitate data entry and visualization of results, providing a more intuitive experience. Implementing this interface would involve software development and design techniques, focusing on system architecture and integration with genetic algorithms. Usability studies would be crucial to ensure the interface meets users' needs, including collecting feedback and testing the interface for continuous improvement. The interface would also allow system customization, offering specific configurations, viewing different perspectives of timetables, and generating personalized reports.

References

Andrade, C. E., Toso, R. F., Gonçalves, J. F., and Resende, M. G. (2021). The Multi-parent Biased Random-Key Genetic Algorithm with implicit Path-Relinking and its real-world applications. *European Journal of Operational Research*, 289(1):17–30.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160.

Bettinelli, A., Cacchiani, V., Roberti, R., and Toth, P. (2015). An overview of curriculum-based course timetabling. *Top*, 23:313–349.

Fonseca, G. H., Santos, H. G., Carrano, E. G., and Stidsen, T. J. (2017). Integer programming techniques for educational timetabling. *European Journal of Operational Research*, 262(1):28–39.

Frinhani, R. d. M. D., Lacerda, R. M., Silva, R. M. A., and Mateus, G. R. (2015). BRKGA para auto-parametrização do GRASP com Path-Relinking no Agrupamento de Dados. In *Anais do XLVII SBPO (Simpósio Brasileiro de Pesquisa Operacional), 25 a 28 de Agosto, Porto de Galinhas/PE*, p. 1780–1792. Sociedade Brasileira de Pesquisa Operacional (SOBRAPO).

Gonçalves, J. F. and Resende, M. G. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press. ISBN 9780262275552. URL `https://doi.org/10.7551/mitpress/1090.001.0001`.

Minh, K. N. T. T., Thanh, N. D. T., Trang, K. T., and Hue, N. T. T. (2010). Using Tabu search for solving a High School Timetabling Problem. *Advances in Intelligent Information and Database Systems*, p. 305–313.

Pillay, N. (2014). A survey of School Timetabling research. *Annals of Operations Research*, 218: 261–293.

Raghavjee, R. and Pillay, N. (2015). A genetic algorithm selection perturbative hyper-heuristic for solving the school timetabling problem. *ORiON*, 31(1):39–60.

Tan, J. S., Goh, S. L., Kendall, G., and Sabar, N. R. (2021). A survey of the state-of-the-art of optimisation methodologies in School Timetabling Problems. *Expert Systems with Applications*, 165:113943.

Tassopoulos, I. X. and Beligiannis, G. N. (2012). Solving effectively the school timetabling problem using particle swarm optimization. *Expert Systems with Applications*, 39(5):6029–6040.

Teoh, C. K., Wibowo, A., and Ngadiman, M. S. (2015). Review of state of the art for metaheuristic techniques in academic scheduling problems. *Artificial Intelligence Review*, 44:1–21.

Willemen, R. (2002). *School timetable construction: algorithms and complexity*. PhD thesis, Mathematics and Computer Science.

Zhang, D., Liu, Y., M'Hallah, R., and Leung, S. C. (2010). A Simulated Annealing with a new neighborhood structure based algorithm for high School Timetabling Problems. *European Journal of Operational Research*, 203(3):550–558.