# Overview of Genetic Algorithms in Educational Timetabling

Luca Quaer
luca@quaer.net
Wilhelm Büchner Hochschule
Darmstadt, Baden-Württemberg, Germany

## ABSTRACT

Empty

## KEYWORDS

Genetic Algorithms, Educational Timetabling, Metaheuristics

## 1 INTRODUCTION

Educational timetabling involves creating schedules for educational institutions such as schools, colleges, and universities. The problem domain can be divided into the following three main problems [7, 9]: High-School Timetabling (HTT), University Course Timetabling (CTT) and University Examination Timetabling (ETT). Although a clear distinction between these three problems is not always possible, they generally differ significantly from one another [2]. However, each of these problems essentially is a resource allocation problem with the goal of assigning classrooms, instructors, and students to specific time slots for various courses or activities, ensuring that all constraints and requirements are met. This includes avoiding conflicts (e.g., a student being scheduled for two classes at the same time), adhering to institutional policies, and maximizing the efficient use of resources.

The difficulty in finding a valid and effective solution to such a problem lies in meeting the diverse requirements of different stakeholders (e.g. students, teachers, administration), multiple constraints and resolving resource conflicts in a combinatorial complex solution space caused by the numerous constraints. Timetabling problems like these are therefore known to be NP-complete in their general form, meaning that the difficulty of finding a solution increases exponentially with the problem size, which in turn makes it impossible to find a deterministic algorithm providing an acceptable solution in polynomial time [2]. One popular approach to addressing the complexity of timetabling problems is the use of metaheuristic algorithms [2]. This class of algorithms leverages a non-deterministic search approach which compromises on finding an optimal solution in favor of better runtime performance. Consequently, such algorithms are not guaranteed to find the best solution for a given problem, but a near optimal one [1]. Despite this limitation, metaheuristic algorithms are widely used in educational timetabling due to their ability to provide high-quality solutions within a reasonable timeframe. These algorithms can be broadly classified into two categories: single-solution and population-based metaheuristics [6]. Single-solution based algorithms use a single candidate solution and iteratively improve it by using local search, but are prone to get stuck in local maxima [6]. Population-based metaheuristics on the other hand work on multiple candidate solutions during the search process, avoiding the risk of getting stuck in a local maximum by maintaining diversity among the solution candidates [6]. Popular single-solution based algorithms in the timetabling domain are simulated annealing, local search and Tabu search [4, 6]. Well-known population based metaheuristics are genetic algorithms, particle swarm optimization and ant colony systems [2, 6].

Among these methods, genetic algorithms are known for their versatility and application in a variety of use cases with the need of searching for solutions of a combinatorial problem in a large solution space. Therefore, this paper specifically focuses on genetic algorithms and how they are used in the domain of educational timetabling.

Genetic algorithms (abbr. *GA*) are a heuristic search method inspired by the process of natural selection in biological evolution and thus belong to the group of evolutionary algorithms [6]. As mentioned previously, genetic algorithms utilize a population based approach, meaning multiple solution candidates are iteratively evolved through numerous generations imitating the Darwinian theory of survival of the fittest [6].

## 2 METHODS

This paper specifically focuses on the application of genetic algorithms in the domain of educational timetabling and not timetabling or scheduling problems in general. To accomplish this, a thorough search among recent and early publications on this topic has been conducted, to create a representative overview of the most important concepts of genetic search used in the field of timetabling. After an introduction of the basic and some advanced concepts of genetic algorithms, the research results will be visualized and discussed.

## 3 BASIC CONCEPTS

Genetic algorithms are a type of search and optimization algorithm inspired by the biological process of reproduction and natural selection and represent one branch in the field of evolutionary computing [3, 5].

In the search for a solution to an optimization problem, the set of possible solutions – the so-called solution space – must first be determined and made comprehensible for an algorithm in form of a data structure, which is suitable for representing a solution [1]. This representation of the solution is also called *encoding* and contains the data of a possible solution to the problem to be solved. In nature, this data is encoded on chromosomes. Similarly, in genetic algorithms the possible solution in coded form is also called *chromosome* or *individual* [1].

In addition, genetic algorithms employ a population based search approach, whereby instead of a single potential solution a whole set of solutions is iteratively improved. Such a set of solutions is called *population* and consists of multiple chromosomes. The stages of iterative improvements are called *generations* [1].

In order for the algorithm to optimize towards a desired solution, it is necessary to have a measure in place to evaluate and compare the chromosomes. This value referred to as *fitness* and is provided by the *fitness function* (also called *objective function*) [1].

With these basic terms defined, the general process of genetic algorithms can now be described as follows: First, an *initial population* must be created and the fitness of its chromosomes must be evaluated [1]. Pairs (or triples, quadruples, etc.) are then selected (*selection* phase) from this population in order to reproduce (known as *crossover*) [1]. The resulting offspring may undergo one or more mutations (*mutation* phase) with a defined probability before the fitness of these new chromosomes is determined (*evaluation* phase) [1]. Based on certain criteria chromosomes from the current generation and their offspring are now selected, to replace the population with a new one (*replacement* phase) [1]. The result of this step is a new generation of chromosomes forming a new (usually fitter) population [1]. From this population, chromosomes are once again selected for reproduction, starting the process all over again [1]. The genetic algorithm could theoretically continue indefinitely according to this pattern, with termination conditions serving as the only means of halting the process [2]. The forementioned phases of genetic algorithms will be explained in the following chapters.

## 3.1 Encoding

Genetic algorithms require two essential components: an encoding and a fitness function [1]. The encoding plays a pivotal role in the design of a genetic algorithm [6]. Its most significant property is that it can completely represent the solution space of the problem at hand, thereby deriving the potential solution to the problem from a given chromosome [1]. Moreover, the encoding must be designed in consideration of the data processing of other genetic algorithm components, such as the fitness function and the crossover operator [1]. The fitness function must calculate the fitness value based on this representation of a solution candidate, and the crossover operator should generate offspring that are as valid as possible [1]. In particular, the latter aspect can often only be fulfilled by an adapted, domain-specific representation [1]. The following paragraphs present some well-known encodings.

*3.1.1 Binary Encoding.* Binary encoding is a method in which chromosomes are represented as strings of binary digits, i.e. an array of 0s and 1s [6]. Each unit of information (also called a *gene*) corresponds to one binary digit (*bit*). The main advantage of binary encoding is the ability to use common and well-researched crossover and mutation operators [6]. However, these strategies lead to invalid representations, which would need to be repaired. Such repair strategies are used in practice but pose the risk of introducing too much genetic information which is from neither of the parents [1]. Furthermore, utilizing this encoding requires converting solution candidates into binary form [6]. Depending on the complexity of the problem this might not be feasible and thus require a different encoding.
Other well-known encoding schemes are *decimal* and *hexadecimal* encoding. They work analogous to binary encoding except for using decimal and hexadecimal digits respectively [6].

*3.1.2 Value Encoding.* Value encoding is similar to binary encoding, as it also represents chromosomes as strings of values. In contrast to binary encoding, these values can be floating point numbers, integers or characters [6]. This encoding scheme is mainly used for finding the optimal weights in a neural network [6].

*3.1.3 Permutation Encoding.* The permutation encoding method is commonly used in ordering problems [6]. Similar to the encodings mentioned before, a chromosome is made up of an array of values, but as the name suggests, the position in the array encodes the order of the values in the context of an ordering problem [1, 6]. Given that the objects of the ordering problem are unique, this implies that the values in the array are unique too [1].

*3.1.4 Matrix Encoding.* Matrix encoding is a technique used in genetic algorithms where solutions are represented as matrices (two-dimensional arrays) rather than as one-dimensional arrays. This encoding is particularly useful for problems that naturally map to a two-dimensional structure, such as scheduling, graphs (as adjacency matrix) layout design, or certain combinatorial problems [1].

## 3.2 Fitness Function

"In genetic algorithms a fitness function assigns a score to each individual in a population; this fitness value indicates the quality of the solution represented by the individual" [1]. "Evaluating the fitness function for each individual should be relatively fast due to the number of times it will be invoked" [1]. Consequently, it is arguably the most crucial component of a genetic algorithm, and it is the only chance to steer the process of genetic evolution in accordance with the desired optimization intentions [2, 8]. Especially in the context of constrained optimization problems with multiple objectives (e.g. hard and soft constraints), the fitness function must be carefully designed to convey the correct optimization target for solving the problem at hand [2, 3]. Timetabling problems usually pose multiple objectives in form of hard and soft constraints [2]. These differ in their severity: hard constraints must be satisfied in order for the solution to represent a valid timetable, soft constraints on the other hand must not be satisfied and only contribute to the quality of the solution [2].
As previously stated, the design of the fitness function heavily depends on the employed encoding scheme. The encoded data of a solution candidate serves as the input value utilized by the fitness function to calculate the fitness of a chromosome [1]. Although there is no universal recipe for designing fitness functions, because they are highly domain-specific, there are some methods that can be used as a starting point. A prevalent method is to utilize a weighted sum as the foundation of a multi-objective fitness function. Furthermore, such a function can be enhanced through the incorporation of a penalty function, which accounts for constraint violations at a specific weight per constraint [2].

## 3.3 Initial Population

Once an appropriate encoding and a fitness function have been established, the initial step in the actual execution of the algorithm is to generate an initial population [1]. The initial population can be generated in two ways: randomly or heuristically. If the problem's difficulty lies not in finding valid solutions, but in finding the optimal solution, then heuristic initialization may be a beneficial approach, as it could facilitate the evolution of the population. However, in cases where finding a valid solution is already a significant challenge, heuristics are not a viable option. In such instances, the initial population must be created randomly [1].

### 3.4 Selection

In the selection phase of a genetic algorithm, chromosomes are selected for mating (*crossover*) [1]. The prerequisite for this phase of the genetic algorithm is, that the fitness of each chromosome in the population has been evaluated [1]. At present, the most commonly employed selection techniques are roulette wheel, rank, tournament, Boltzmann, and stochastic universal sampling [6]. The following sections present a brief overview of selected methods.

*3.4.1 Roulette Wheel.* See [1] and [6].

*3.4.2 Tournament.* See [1] and [6].

*3.4.3 Rank.* See [1].

*3.4.4 Comparison of selection techniques.* See table 3 in [6]

### 3.5 Crossover

Crossover is the genetic operator which performs the actual mating of individuals selected during the selection phase [2]. Given the close relation between the encoding scheme and the crossover operator, certain crossover operators may not be compatible with some encodings in context of the problem at hand [1].

*3.5.1 Single Point.* See [1].

*3.5.2 Multiple Point.* See [1] and [6].

*3.5.3 Uniform Point.* See [1].

*3.5.4 Partially matched crossover.* See [6].

*3.5.5 Order crossover.* See [6].

*3.5.6 Shuffle crossover.* See [6].

*3.5.7 Comparison of different crossover techniques.* See table 4 [6].

### 3.6 Mutation

The mutation operator is used on the offspring created by the crossover operation, to allow undirected jumps to slightly different areas of the search space [1]. This procedure maintains genetic diversity throughout generations and helps in efficiently exploring the search space [1, 6].
The actual implementation of the mutation operator greatly depends on the chosen encoding, because mutating a chromosome could potentially lead to an invalid solution candidate depending on how the encoding is designed.
Well-known mutation operators are displacement, simple inversion and scramble mutation.

*3.6.1 Displacement Mutation (DM).* See [6].

*3.6.2 Simple Inversion Mutation (SIM).* See [6].

*3.6.3 Scramble Mutation (SM).* See [6].

### 3.7 Evaluation

For the sake of completeness, the evaluation is listed here as separate phase, even though this is not common in the researched literature. The evaluation phase serves as the final step after successful application of the crossover and mutation operations, in which the fitness of the new individuals must be calculated in order to continue with the genetic algorithm.

### 3.8 Replacement

After the current generation has reproduced in the selection, crossover and mutation phase, which created new offspring, the question arises as to which of the new solution candidates should become members of the next generation [1]. In context of evolution the replacement strategy determines the life span of the individuals and substantially influences the convergence behavior of the algorithm [1]. The following schemes are possible replacement strategies for genetic algorithms:

*3.8.1 Generational Replacement.* [1]

*3.8.2 Elitism.* [1]

*3.8.3 Delete-n-last.* [1]

*3.8.4 Delete-n.* [1]

*3.8.5 Tournament Replacement.* [1]

### 3.9 Termination

As previously stated, the evolutionary process in genetic algorithms is an infinite loop that requires a termination criterion to halt. The desired termination constraint may vary depending on the problem and the context in which the algorithm is used in. One straightforward approach is to simply stop the genetic algorithm after a certain number of generations has been reached [2]. Another widely used method is to terminate the algorithm when the fitness value of the best individual has not changed over a predefined number of generations [3].

## 4 ADVANCED TECHNIQUES

### 4.1 Direct and Indirect Encoding

### 4.2 Non-standard genetic operators

### 4.3 Offspring Selection (OS)

### 4.4 Selection Pressure

### 4.5 Parallel Genetic Algorithms

## 5 DISCUSSION

To do.

## 6 CONCLUSION

To do.

## 7 FUTURE WORK

The findings from this work will serve as the basis for the author's master's thesis, in which a genetic algorithm for a special timetabling problem will be developed. The examinations conducted in this paper provide information about the most frequently used techniques of genetic search within the timetabling domain, which will serve as a reference point for selecting the most appropriate genetic methods for the algorithm to be developed.

# REFERENCES


[1] Michael Affenzeller, Stephan Winkler, Stefan Wagner, and Andreas Beham. 2009. Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications.

[2] G. N. Beligiannis, C. Moschopoulos, and S. D. Likothanassis. 2009. A genetic algorithm approach to school timetabling. *Journal of the Operational Research Society* 60, 23–42. Issue 1. https://doi.org/10.1057/palgrave.jors.2602525

[3] Jenna Carr. 2014. An Introduction to Genetic Algorithms.

[4] Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. 2023. Educational timetabling: Problems, benchmarks, and state-of-the-art results. , 18 pages. Issue 1. https://doi.org/10.1016/j.ejor.2022.07.011

[5] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*. https://doi.org/10.1023/A:1022602019183

[6] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. 2021. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications* 80 (2 2021), 8091–8126. Issue 5. https://doi.org/10.1007/s11042-020-10139-6

[7] Jeffrey H Kingston. 2013. Educational timetabling. In *Automated Scheduling and Planning: From Theory to Practice*. Springer, 91–108.

[8] Kenneth E Kinnear Jr. 1994. A perspective on the work in this book. *Advances in genetic programming* (1994), 3–19.

[9] Andrea Schaerf. 1999. A survey of automated timetabling. *Artificial intelligence review* 13 (1999), 87–127.