Overview of Genetic Algorithms in Educational Timetabling

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

ABSTRACT

Todo!

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 [11, 13]: 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 [4]. 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 [4]. One popular approach to addressing the complexity of timetabling problems is the use of metaheuristic algorithms [4]. 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 [10]. 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 [10]. 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 [10]. Popular single-solution based algorithms in the timetabling domain are simulated annealing, local search and Tabu search [6, 10]. Well-known population based metaheuristics are genetic algorithms, particle swarm optimization and ant colony systems [4, 10].

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 [10]. 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 [10].

2 METHODS

Veröffentlichungen der jüngeren Vergangenheit, insbesondere Survey Paper in diesem Feld, welche Lösungsansätze für educational timetabling behandeln, zeigten, dass die Populatität von Populationsbasierte Algorithmen wie genetische Algorithmen abgenommen hat. Dadurch hat sich im Bereich der genetischen Algorithmen eine Forschungslücke aufgetan. Außerdem ist dies kein "starker Indikator" dafür, dass diese Art von Algorithmen grundsätzlich nicht geeignet für Timetabling Probleme ist, denn in diesem Umfeld werden oft einfach die Algorithmen angewand, mit denen die jeweilige Forschungsgruppe bereits Erfahrung gesammelt hat.

Das Ziel dieser Arbeit ist, die bisherige Verwendung von genetischen Algorithmen speziell im Bereich des Educational Timetablings aufzuzeigen und übersichtlich darzustellen. Grundlage dessen ist eine umfassende systematische Literaturrecherche die durchgeführt wurde, um viele relevante Anwendungen von genetischen Algorithmen in der Domäne des Edu. Timetabling zu finden und anhand wesentlicher Eigenschaften einzuordnen. Dabei wurde auf verschiedene wissenschaftliche Datenbanken wie IEEE Xplore, ACM Digital Library, SpringerLink und Google Scholar zurückgegriffen. Die Suchkriterien umfassten Schlüsselbegriffe wie "genetic algorithm", "educational timetabling", "timetabling", "scheduling", "heuristic" and "metaheuristic". Es wurde darauf geachtet, dass vorzugsweise aktuelle Veröffentlichungen, jedoch auch ältere, etablierte Studien und Paper berücksicktigt wurden, um einen umfassenden Überblick über die Entwicklung und den Einsatz genetischer Algorithmen in diesem Feld zu erhalten und keine bereits eingesetzten Methoden aufgrund ihres Alters auszuschließen. Hauptaugenmerk war nichts desto trotz die ausführliche Beschreibung des Algorithmus, um dessen Eigenschaften möglichst ausführlich

zu dokumentieren. Dieses Kriterium ist in vielen Veröffentlichungen nicht gegeben, vor allem der Algorithmus bewusst nicht veröffentlicht wurde und deshalb nur auf dessen Ergebnisse eingegangen wird.
Die Gliederung der vorliegenden Arbeit ist wie folgt: Zunächst werden die Grundlagen von genetischen Algorithmen und zusätzlich ausgewählte erweiterte Konzepte behandelt. Diese Einführung dient der besseren Einordnung der später vorgestellten Algorithmen und deren Eigenschaften. Die vorgestellten Methoden sind und können nicht erschöpfend sein und das breite Feld der genetischen Algorithmen vollumfänglich behandeln.
Danach werden die in der Literaturrecherche herausgesuchten Algorithmen anhand eines morphologischen Kastens genannt und deren wesentliche Eigenschaften erarbeitet. Dies ermöglicht einerseits eine Übersicht über den aktuellen Stand der Forschung bezüglich genetischer Algorithmen für Educational Timetabling und zeigt zusätzlich neue Forschungsansätze und -möglichkeiten auf, welche sich durch nicht verwendete Kombinationen aus den Eigenschaften der Algorithmen ergeben. Durch die systematische Analyse und Darstellung im morphologischen Kasten sollen Wissenschaftler und Praktiker inspiriert werden, innovative und effektive Lösungsansätze zu entwickeln und bestehende Methoden weiterzuentwickeln.
Abschließend wird über übergeordnete Erkentnisse aus der Literaturrecherche eine Aussicht auf den potentiellen Erfolg von genetischen Algorithen in zukünftiger Forschung gegeben.

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 [5, 8].
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 [4]. 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 [10]. 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 [10]. 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 [10]. However, these strategies lead to invalid representations, which would need to be repaired [4]. 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, 4]. Furthermore, utilizing this encoding requires converting solution candidates into binary form [10]. 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 [10].

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 [10]. This encoding scheme is mainly used for finding the optimal weights in a neural network [10].

3.1.3 Permutation Encoding. The permutation encoding method is commonly used in ordering problems [10]. 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, 10].

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 [4, 12]. 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 [4, 5]. Timetabling problems usually pose multiple objectives in form of hard and soft constraints [4]. 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 [4].
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 [4].

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 [10]. The following sections present a brief overview of selected methods.

3.4.1 Roulette Wheel. The roulette wheel selection method is also called proportional selection and works by assigning each individual a probability for reproduction according to its fitness relative to the whole population [1]. The expected value of the i-th chromosome to be selected for reproduction is therefore $p_i = \frac{f_i}{\overline{f}}$ with f_i denoting the fitness of the i-th chromosome and $\overline{f}$ representing the average fitness of all individuals in the population [1]. "Therefore, each individual of the population is represented by a space proportional to its fitness"[1] on an imaginary roulette wheel. "This wheel is then rotated randomly to select specific solutions that will participate in formation of the next generatoin"[10].

3.4.2 Rank Selection. "Rank selection is the modified form of Roulette Wheel selection. It utilizes the ranks instead of fitness value". See [1]. A possible approach of realizing this selection method is by ordering the individuals according to their fitness and add copies of individuals in a way that the best individual receives a pre-determined multiple number of copies the worst one receives [1].
Using this selection strategy reduces the dominating effect of outstanding fit individual, but in turn distorts the difference between similarly fit individuals, which increases the selection pressure in stagnant populations [1]. Compared to Roulette Wheel selection it reduces the chances of premature convergence towards a local optimum [10].

3.4.3 Tournament Selection. There are numerous variants of this selection method [1]. The most common one is k-tournament selection where k individuals are randomly selected from a population and the fittest individual of the selected ones is considered for reproduction [1]. By choosing the tournament size k appropriately, the selection pressure can be easily scaled [1].

3.5 Crossover

Crossover is the genetic operator which performs the actual mating of individuals selected during the selection phase [4]. 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. The single point crossover method cuts each parent chromosomes into a head and tail section. For this a random position is chosen at which both chromosomes are cut [1]. For a chromosome using binary encoding for example, this translates to splitting the array used to represent the chromosome into two sections. The tail sections are then swapped producing two new individuals [1].

3.5.2 Multiple Point. Similar to the single point crossover method, the multipoint crossover cuts the parent chromosomes into sections. In contrast to the single point crossover, and as the name suggests, the multipoint crossover method uses multiple random cuts instead of a single one [1, 10].

3.5.3 Uniform Crossover. Uniform crossover uses a randomly generated crossover mask, which contains information about what parent chromosome provides certain genes to the offspring [1]. This information could for example be: "the first and second gene (e.g. first and second value in the array) are taken from the first parent, the third gene is taken from the second parent". To yield two children from the crossover operation, the mask can be inverted. This process is equal to randomly swapping genes (at the same position) between the parents and taking the resulting chromosomes as offspring [10].

3.5.4 Partially matched crossover. Partially matched crossover is the most commonly used crossover operator, as it performs better than most of the other operators [10]. This crossover technique works by randomly choosing a continuous part of genes from one parent, and swapping it with the corresponding genes (at the same position) from the other parent chromosome [10]. The left out genes are simply copied to the offspring chromosomes, except of genes with values which are contained in the parts that were swapped between the parents. The swapped section of genes denotes a mapping which is used to map these values to the offspring chromosomes instead of simply copying them [10].

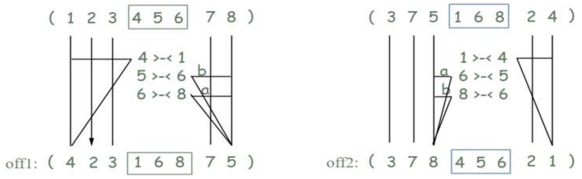

Figure 1: Partially matched crossover visualization [10]

3.5.5 Order crossover. Order crossover copies one (or more) parts of the parent to the offspring and fills the remaining space with genes from the respective present, which are not present in the copied section [10].

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, 10].
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). The displacement mutation operator selects a random section from a chromosome and moves it to another position on the chromosome [10]. This operation does not change the sequence of the genes not included in the moved section.

3.6.2 Simple Inversion Mutation (SIM). For simple inversion mutation a random section is selected from a chromosome. The order of the genes in this section is then reversed [10]. This mutation can be further enhanced by also moving the section to another position within the chromosome (similar to displacement mutation) [10].

3.6.3 Scramble Mutation (SM). The scramble mutation is very similar to the simple inversion mutation with the single difference of not inverting but shuffling the genes in the randomly selected part of the chromosome [10].

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] As the name already suggests, the generational replacement strategy simply replaces the current population with the newly created offspring. This process may result in a decrease of the fitness of the fittest individual in the population at some stages of evolution [1].

3.8.2 Elitism. With an elitism method of generation replacement, the best individual (or the n best individuals) of the previous generation are retained for the next generation [1, 10]. This theoretically allows individuals to be immortal and might lead to premature convergence [1]. The special case of only retaining the best individual is also called "golden cage model" (n-elitism with $n = 1$) [1]. In case the mutation operator is applied to the elite individuals to prevent premature convergence, the replacement strategy is called "weak elitism" [1].

3.8.3 Delete-n-last. The n weakest individuals are replaced by n descendants [1]. If n is much smaller than the population size, this strategy is known as steady-state replacement scheme [1]. For $n = 1$, the changes between the old and new generations are minimal, while choosing n equal to the size of the population represents the previously discussed generational replacement strategy [1].

3.8.4 Delete-n. In contrast to the delete-n-last replacement strategy, this approach replaces n arbitrarily chosen individuals from the old generation rather than the weakest ones [1]. While this reduces the convergence speed of the algorithm, it also helps to avoid premature convergence, balancing between elitism and weak elitism [1].

3.8.5 Tournament Replacement. Tournament replacement is similar to the equally named selection strategy. In this replacement scheme competitions are run between sets of individuals from the old population and their offspring [1]. The winners of these tournaments become part of the new population [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 [4]. 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 [5].

4 ADVANCED TECHNIQUES

In the field of genetic programming, it is common practice to adapt existing methods to fit one's own use case. This has led to a steady development of new approaches to improve genetic search in general. While it is beyond the scope of this paper to provide an exhaustive overview of the many branches of advanced genetic search, the following sections present a selection of important in regard to solving timetabling problems.

4.1 Direct and Indirect Encoding

Genetic algorithms can be classified into two primary categories based on the employed encoding scheme: *direct* and *indirect* [14]. In a direct encoding, the chromosome encodes all features of a solution candidate, meaning that the whole search space is encoded [9, 14]. Depending on the problem domain, this means, that chromosomes may also represent invalid solution candidates. Therefore, direct encodings are prone to hard constraint violations on crossover and mutation operations, which requires applying additional mechanisms to find feasible solutions [9]. One possible solution is to repair these invalid chromosomes by applying a repair function that uses domain-specific heuristics to transform the chromosome to a valid state, with the downside of introducing genetic information not related to either of the parents [1, 4].

In contrast to a direct representation, an indirect (or *implicit*) encoding only *partially* encodes a solution candidate [14]. One advantage of not encoding all information of a solution is, that this restricts the search space the algorithm has to explore [9]. Furthermore, by using an indirect encoding, compliance with hard (and even soft) constraints can be guaranteed, if the encoding is designed to only represent valid solutions [9].

School timetabling problems are highly constrained, hence indirect encodings have been successfully applied to such problems [9].

4.2 Custom genetic operators

Especially for solving more complex solutions, like creating the forementioned school timetable with many constraints, customized variants of genetic operators may be necessary [2, 4]. An example for such a custom operator is presented in [4]. Their crossover operator is derived from a typical uniform crossover, but it is adapted to the problem domain to not cause problems with a teacher's schedule [4]. Another potential application of such custom genetic operators could also be the prevention of producing infeasible solutions, when using encoding schemes that do not rule such invalid chromosomes out by design [7]. Similarly to the custom crossover operator mentioned above, mutation operators must also be adapted according to the chosen encoding scheme [2].

4.3 Selection Pressure

In the context of genetic algorithms, selection pressure refers to the intensity with which the algorithm favors the fittest individuals during the selection process for reproduction [1, 3]. When selection pressure is high, the algorithm strongly favors the best individuals in the population. These individuals are more likely to be selected for reproduction, leading to a quicker convergence towards optimal or near-optimal solutions. However, if the pressure is too high, it might cause premature convergence, where the population loses diversity and gets stuck in local optima [3].

Conversely, when selection pressure is low, the algorithm less strongly favors the fittest individuals. This allows for a more diverse set of individuals to be selected for crossover. This helps to maintain genetic diversity within the population and can avoid premature convergence. However, if the pressure is too low, the algorithm may converge very slowly or struggle to find optimal solutions efficiently [3].

Consequently, the selection pressure generated by the utilized selection method is always a compromise between convergence speed and avoiding premature convergence. It appears, that a dynamic intervention in the evolutionary process to adapt the selection process and the associated selection pressure would result in an improvement of the algorithm. This is exactly the approach taken by the offspring selection method explained in the following paragraph.

4.4 Offspring Selection

Offspring selection (OS) is a self-adaptive selection pressure steering method [1]. Contrary to what the name suggests, this method does not replace previously presented selection methods, such as roulette-wheel or linear-rank schemes. Instead, a second selection step is introduced [1]. After the first selection step has been executed (for example roulette-wheel selection) and the crossover operation has been applied to the selected chromosomes, a further selection – the offspring selection – is applied [1].

The offspring selection considers the fitness of the individuals resulting from crossover. To assure the genetic search mainly progresses with successful offspring, the newly introduced selection step guarantees that a sufficient number of children surpass their parents' fitness [1]. The success ratio ($SuccRatio \in [0, 1]$) controls the proportion of individuals in the next generation, with better fitness than their parents [1].

This inevitably leads to the following question: "Is a child better than its parents, if it surpasses the fitness of the weaker parent, the better parent, or some kind of weighted average of both?" [1]. To answer this question the offspring selection method introduces a comparison factor ($CompFactor \in [0, 1]$), inspired by simulated annealing. This factor sets a fitness threshold between the worse and better parent. Early in the process, offspring only need to

surpass the fitness of the weaker parent to be considered *better*. As the algorithm progresses, this threshold increases towards the fitness of the better parent, facilitating a broader search initially and a more focused search later [1]. The gradual increase of the comparison factor adjusts the selection pressure throughout the genetic search process, starting with a low selection pressure to prevent premature convergence and steadily increasing the pressure to not suffer from long runtimes until the search converges [1].

4.5 Parallel Genetic Algorithms

Genetic algorithms are well suited for parallelization. There are various methods of implementing these, some of which require fundamental changes to the algorithm and others do not [1]. Concepts for parallel genetic algorithms fall into three categories: global parallelization, coarse-grained parallel GAs and fine-grained parallel GAs. The most popular for practical applications is the coarse-grained model, also known as the island model [1].

Global Parallelization. In global parallelization, a single population is used, and selection involves all individuals. This method retains the same qualitative properties as a sequential GA, with the primary parallelized operation being the evaluation of individual fitness. A master node distributes and collects workloads from slave processors, making this model efficient when fitness evaluation is the primary runtime bottleneck [1].

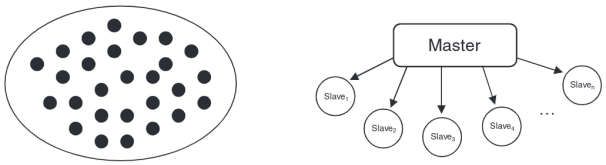

Figure 2: Global parallelization: Single population (left) and the corresponding master-slave model (right) [1].

Coarse-grained parallel genetic algorithms. Coarse-grained parallel GAs divide the population into subpopulations (called *islands* or *demes*) that evolve mostly in isolation, occasionally exchanging individuals during migration phases. This model introduces significant changes to the algorithm structure, differing from sequential genetic algorithms [1]. The main idea is that isolated demes will converge to different regions of the solution space, with migration and recombination combining relevant solution parts [1]. Coarse-grained parallel genetic algorithms are widely used due to their ease of implementation and therefore the most popular method of parallelizing genetic algorithms [1].

Fine-grained parallel genetic algorithms. Fine-grained parallel genetic algorithms involve many small demes which are part of one spatially distributed population [1]. The idea behind this is that individuals are spread throughout the global population like molecules in a diffusion process, with recombination restricted to local neighborhoods [1].

5 DISCUSSION

In diesem Kapitel werden die Ergebnisse der systematischen Literaturanalyse vorgestellt und diskutiert. Der Fokus liegt dabei auf der

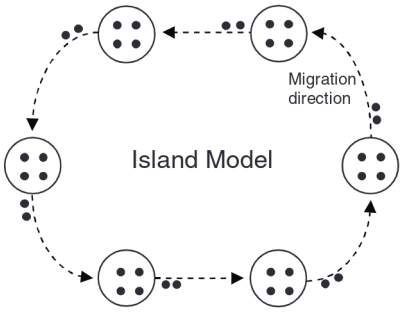

Figure 3: Population structure of a coarse-grained parallel genetic algorithm [1].

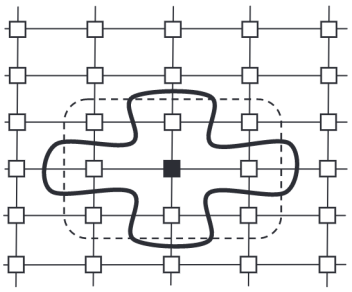

Figure 4: Population structure of a fine-grained parallel genetic algorithm with a cellular model [1].

Darstellung der verschiedenen genetischen Algorithmen und der Darstellung derer Eigenschaften in einem morphologischen Kasten. Dabei wird untersucht, welche Kombinationen von Eigenschaften bisher erfolgreich angewendet wurden, welche bis dato weniger Aufmerksamkeit erfuhren und welche neuen Kombinationen Potenzial für zukünftige Forschung bieten.
Die Tabelle xyz zeigt die im Rahmen der Literaturrecherche betrachteten Algorithmen und deren wesentliche Charakteristika:

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] Maria Weslane Sousa Almeida, Joao Paulo Souza Medeiros, and Patricia Rufino Oliveira. 2015. Solving the academic timetable problem thinking on student needs. *Proceedings - 2015 IEEE 14th International Conference on Machine Learning and Applications, ICMLA 2015*, 673–676. https://doi.org/10.1109/ICMLA.2015.184

[3] Thomas Back. 1994. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the first IEEE conference on evolutionary computation. IEEE World Congress on Computational Intelligence*. IEEE, 57–62.

[4] 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

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

[6] 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

[7] David Graham Elliman. 1995. A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems. http://www.yandy.com

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

[9] G Goos, J Hartmanis, and J Van Leeuwen. 2002. Lecture Notes in Artificial Intelligence Lecture Notes in Computer Science.

[10] 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

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

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

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

[14] Nguyen Duc Thanh. 2007. Solving Timetabling Problem Using Genetic and Heuristic Algorithms. *Proceedings - SNPD 2007: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing* 1, 591–594. https://doi.org/10.1109/SNPD.2007.464