Tabu Search for optimising the timetables of university exams

De Schepper Cedric

Principal Advisor: Prof. Dr. Serge Demeyer
Assistant Advisor: Joey De Pauw

June 2, 2023

Contents

List of Figures

List of Tables

Abstract

English abstract.

Nederlandstalige Samenvatting

Dutch Abstract.

Acknowledgments

Student’s personal acknowledgements.

1 Introduction

Still have to write this section

The task of creating a university exam timetable can be reduced to a scheduling problem. The goal is to assign all exams to available time slots in order to produce a schedule without conflicts. Additionally, the distribution of exams has to be optimised as to provide a student with the highest chance of success. These requirements generally are defined as hard and soft constraints. Hard constraints are requirements that have to be met in order to considered a feasible solution, while soft constraints are preferred to be violated as little as possible. Not all timetabling problems might have feasible solutions depending on the data set use

2 Problem Definition

2.1 Educational timetabling

Educational timetabling problems generally can be divided into three distinct problems, each with their own set of characteristic [2, 3]:

High-School Timetabling (HTT)
The creation of weekly schedules for all the classes of a high-school. This problem focuses on assigning classes and teachers to time slots such that no class or teacher has more than one lecture at the same time and a teacher has the needed amount of time slots to teach all lectures. Classes are considered a fixed set of students, that remains the same for each lecture. This makes that two classes will never have common students. Moreover, room availability is often disregarded by assuming all classes have their own room.

University Course Timetabling (CTT)
The creation of weekly schedules for all courses given at universities. Different from HTT, universities can have common students between courses. This requires student enrolment to be taken into account when creating course schedules. Additionally, room availability and capacity become an important factor to consider.

University Examination Timetabling (ETT)
The creation of a schedule for the exams given for university courses. While very similar to CTT, the main difference lies in the constraints determining the feasibility and quality of the timetable. Most importantly, conflicts have a higher impact and distribution of exams has to be considered.

The timetabling problem, as experienced by the University of Antwerp (UA), falls into the category of ETT.

2.2 Problem complexity

Exam timetabling has been proven to be NP-complete [4], meaning that there is no known algorithm that is able to find an optimal solution in polynomial time. When a problem can be reduced to another NP-complete problem, that problem is considered to be NP-complete as well. This means that the NP-completeness can be explained intuitively by reducing it to the graph colouring problem, which has been proven to be NP-complete [5]. This is done by viewing periods as the colours and exams as nodes. Two nodes are connected by an edge whenever the corresponding exams have a common student. A solution is considered feasible when no connected edges are assigned the same colour and thus period.

Because of the NP-completeness, finding an optimal solution is generally not possible for large problem instances. It follows that approximate methods such as heuristics have to be used to tackle this problem [6]. While heuristics do not guarantee optimal solutions, they attempt to generate good solutions within a reasonable amount of time.

2.3 Constraints and data requirements

For this thesis, we study the generation of examination timetables for two faculties namely the Faculty of Applied Engineering (FTI) and the Faculty of Science (FWET). From discussion with the parties responsible for scheduling the exams, it quickly becomes clear that this is a complex problem due to the amount of students, courses, and exam rooms to be considered. In order to define the feasibility and quality of a solution, the hard and soft constraints,characterising the problem, must be determined. The following hard constraints, used to validate the feasibility of a solution, were defined together with the faculties' administration:

- Constraint 1: Each exam is scheduled to a time slot.
- Constraint 2: Each exam is scheduled to a room of the correct type. Every exam requires a specific type of room that must be respected. Such room types can includes seminar rooms, auditoriums, computer rooms, and practicums
- Constraint 3: The number of students enrolled in an exam cannot exceed the capacity of the scheduled room.
- Constraint 4: Each exam is scheduled to a room available to the faculty. All faculties have access to their own set of rooms, with overlap between the faculties possible.
- Constraint 5: No student must be scheduled to more than 1 exam on the same day.

Additionally, the soft constraints, used to determine the quality of the solution, are:

- Constraint 6: The number of days between exams must be maximised

Constraint 1 to 5 are considered to be hard constraints with constraint 6 being the sole soft constraint. Formally this means that the first five constraints must be satisfied in order to provide a feasible solution, with constraint 6 being taken into account to improve it. However, constraint 5 is considered to be a special case because of the notion of a 'model track' in Belgian universities.

Belgian universities provide students with a pre-configured course schedule they can use to graduate within the minimum time duration. This schedule takes into account the prerequisites needed to enrol in every course. This results in large groups of students to follow the same courses. The university administration prioritises creating an optimal exam distribution for these students, while also attempting to take into account the custom tracks as much as possible. This means that constraint 5 can be relaxed to only apply to students following the model track. In further sections, we refer to the two constraint variations as the following:

- Constraint 5a: No student must be scheduled to more than 1 exam on the same day.
- Constraint 5b: No student following the model track must be scheduled to more than 1 exam on the same day.

In order to solve this scheduling problem, some data must be provided or generated:

Students
The set of all students that are enrolled in exams and will impact the timetabling.

Exams
The set of all exams that must be scheduled. Every exam has its own set of students that are enrolled. This allows us to verify if two exams have common students between them and will create a conflict if they occur on the same day. Additionally, an exam belongs to a faculty and is of a certain type. Every exam type requires an allowed room type. For example, computer exams must be held in a computer room.

Rooms
The set of rooms present for examinations. Each room has a room type, a student capacity and a list of faculties it is available for. An exam can only be scheduled in a room if the capacity is sufficient, the type of exam is possible within the room, and the room is available to the faculty to which the exam belongs.

Time slots
The set of all time slots that can be used to schedule exams. We consider a time slot to be determined by its date, exam time, and room. The exam time determines the amount of times a room can be used per day.

Periods
Set of periods, one for each date within the examination schedule. Each period contains all time slots of that date. By checking all students that are enrolled within exams scheduled to a time slot of a period, the amount of constraint 5 violations can be calculated. These periods can be generated by adding a time slot for every room and time combination. The amount of time slots in a period corresponds to $|\text{Rooms}| * |\text{Exam Times}|$.

3 Related Work

This section details different types of algorithms researched to solve the timetabling problem. Additionally, it will also go over several benchmarks used to compare these algorithms.

Several classes for optimisation methods exist, all with their own unique characteristics. Algorithms from the following classes are covered in this section:

Single Solution-based Meta-heuristics

Single Solution-based Meta-heuristics are also commonly referred to as Local Search (LS) methods [7]. These heuristics first generate an initial solution. Afterwards, small changes are continuously applied to the solution using a specific strategy in order to traverse the search space.

They can often be divided into either one-stage or two-stage algorithms [8]. One-stage methods attempt to satisfy both the hard and soft constraints during the same stage. Two-stage approaches initially only take the hard constraints into account in order to generate a feasible solution. Afterwards, the second phase attempts to optimise the solution by adding the soft constraints. Since LS methods only accept a solution when its better than the previous one, encountering a worse solution blocks it from progressing. This means that LS requires strategies in order to escape local optima to keep improving the solution.

Population-based Meta-heuristics

Instead of applying meta-heuristic on a single solution, Population-based meta-heuristics maintain a set of solutions, called a population. Every iteration, a new population is generated by applying meta-heuristics. These methods offer the advantage over single solution-based methods that multiple solutions are presented during the final iteration. Another difference is that they prioritise exploration of the search space while single solution-based methods focus more in exploitation of their solution [9]. However, this exploration does come at the cost of an increase in run time due to a larger amount of operations applied at each iteration.

Exact Methods

Exact methods constrain the problem by defining a linear objective function and force some or all parameters to be integers. Since these methods make use of an exhaustive search process, the search space must be reduced as much as possible to reduce the run time. Unlike heuristics, exact methods can provide upper and lower bounds of the objective function and thus provide proof of optimality. Fundamentally, some or all required variables are constrained to integers.

Hyper-heuristics

Hyper-heuristics [10, 11] are designed to be problem-independent meaning no domain expertise and customisation is needed to be successfully applied to different problems. They utilise the heuristic the most optimal for the given problem.

3.1 Standardised data sets and benchmarks

Even though the timetabling problem has been tackled in many research papers, Ceschia et al. [12] state that many of the initial benchmarks are not relevant anymore. They observe that many papers do not provide the data, source code or solutions used. This makes it impossible to reproduce the presented results. Additionally, many of the problems researched are unique to specific institutions, reducing the amount of relevant observations that can be made from the results. However, effort has been put into standardising the timetabling problems into data sets that can be used to accurately compare different heuristics. The agreement regarding the need for benchmarks dates back to the first International Conference on the Practice and Theory of Automated Timetabling (PATAT) in 1995 [13]. Most of the current benchmarks available come from competitions like the International Timetabling Competition (ITC).

An early formulation for university exam timetabling was done by Carter et al. [14]. They initially introduced 13 exam timetabling data sets for several institutions, commonly referred to as the Toronto or Carter benchmarks. Additional instances have been made available over time [15]. The data behind this format is implemented as a binary enrolment matrix, showing for each exam and student whether the student is enrolled for that exam. The objective function to compare solutions for this benchmark is based on the distance between exams and the amount of common students. The penalty for two exams with k common students at a distance of i periods was calculated as $k * w_i$. The values for w_1, w_2, w_3, w_4, and w_5 were set as 16, 8, 4, 2, and 1, respectively. While this benchmark is still being used to compare heuristics, the simplification of several common constraints means that it less suitable to apply on real-world scenarios. For example, the benchmark is considered to be uncapacitated. This means that exam rooms are not taken into account when generating timetables. Since constraint 2 determines room capacity to be nonnegotiable in our use case, our timetabling problem does not fit in this format.

A later examination timetabling formulation was proposed for ITC-2007 by McCollum et al. [16]. This formulation is more advanced than the one proposed by Carter et al., taking more constraints into account. New constraints include the addition of exam rooms and more strict penalties related to the distance of scheduled exams. Initially, a data set of 12 instances was released for the competition but more real-world instances have been translated to the format over time [17], as well as the implementation of a generator to create artificial instances [18].

Other common benchmarks focus on either the high school or university course scheduling problem. For example, Post et al. [19] were the first to propose a standardised format for high school timetabling. They developed XHSTT, an XML format to be used in solution benchmarks. Several data sets have been made available at `https://www.utwente.nl/en/eemcs/dmmp/hstt/`. Additionally, this format has been used as the target for the third International Timetabling Competition in 2011. Unfortunately, university exam timetabling problems do not allow to be converted into course timetabling formats due to differences in constraints and schedule characteristics. Other examples include the Curriculum-based Course Timetabling (CB-CTT) [20] and Post-Enrolment Course Timetabling (PE-CTT) [21] formats, both used for ITC-2007.

These problem formulations provide valuable contributions in standardising the timetabling research, allowing for easier comparison between algorithms and checking of solutions. However, the uniqueness of the use case for the University of Antwerp makes using these benchmarks infeasible. As a result, the comparison of the quality of generated solutions can only be made against those produced manually by the university's administration.

3.2 Single Solution-based Meta-heuristics

Single Solution-based Meta-heuristics or Local Search methods start from an initial solution. Generally, this solution is generated randomly or using a greedy algorithm, but other techniques can also be used. LS then starts exploring the search space by calculating the neighbourhood of its current solution and picking a neighbour as its new solution. Different versions of Local Search use their own technique for choosing the neighbour and when to stop the search. The objective function is used to estimate the quality of each solution, with LS attempting to miminise this objective.

3.2.1 Tabu Search

Tabu Search (TS) [22] is based on Local Search (LS) [7], meaning it traverses the possible search space by performing local changes to the current solution. Since LS will only accept improving solutions, this traversal can end up being stuck in a local optimum. TS differs from LS by relaxing this rule and also accepting worsening solutions. This relaxation allows for more exploration and allows TS to escape local optima. Additionally, TS maintains a memory structure to avoid changes being reversed. The usage of short- to long-term memory is based on the assumption that optimisation techniques must incorporate memory to qualify as intelligent and that a bad strategic choice is superior to a good random choice [23]. This memory structure is implemented by maintaining a tabu list which contains the x most recent changes performed. A change is thus considered 'tabu' if it is present within the tabu list.

Alvarez-Valdes et al. [24] proposes a two-phased Tabu Search in order to solve the university exam timetabling problem. During the first phase, the emphasis is on satisfying the hard constraints. More specifically, they attempt to reduce the amount of occurrences where a student has two exams on the same day. Optimally, this initialisation phase would return a solution that is already feasible. The second phase can be considered an optimisation phase, which tries to satisfy the soft constraints as much as possible and thus attempt to spread the exams as best as possible. They used this algorithm on real-world data sets of the University of Valencia, and compared it with the manually designed exam schedules. They conclude that the generated schedules are superior for all data sets.

Di Gaspero and Schaerf [25] adapt the original TS method by proposing changes to the tabu list and stopping criterion. Instead of keeping a fixed tabu list size, each move is assigned a number of iterations that the move is considered tabu. Additionally, the algorithm will terminate after a fixed amount of iterations without improvement. When comparing their results with the Carter benchmarks, they note that Tabu Search provides comparable performance with the benchmarks.

Chu and Fang [26] made a comparison between using Genetic Algorithms versus Tabu Search to obtain time tables. For all of their different experiments, the Tabu Search (TS) implementation was able to outperform their GA version, both on the quality of the solution and the computational time needed to converge. However, a redeeming quality of Genetic Algorithms is that they are able to produce several near optimal solutions in one go while Tabu Search implementations are limited to a single solution.

Colorni et al. [27] apply Tabu Search, Simulated Annealing, and Genetic Algorithms on the high school timetabling problem for an Italian high school. They conclude that TS outperforms both competitors in generating quality timetables for their use case.

3.2.2 Simulated Annealing

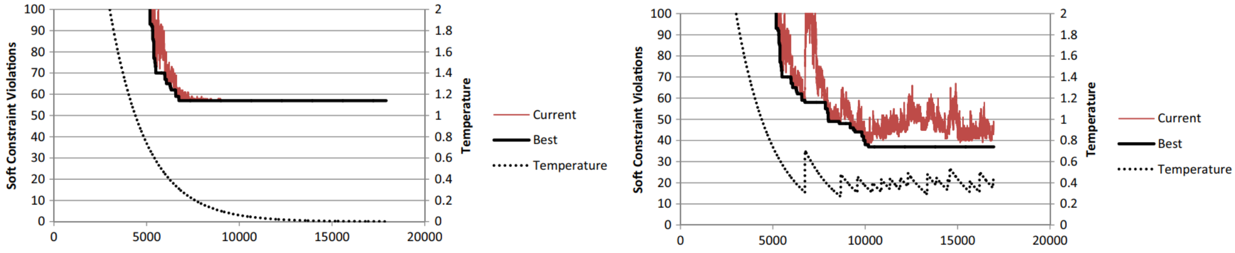
Aycan and Ayav [28] apply Simulated Annealing (SA) [29] to generate optimal solutions. In order to create a initial solution as optimal as possible, constraint satisfaction methods are used to create a solution satisfying all hard constraints. During the simulated annealing phase, a new solution is found by randomly swapping two variables. The cost of the new solution is then calculated using the objective function. Whenever the new objective is lower than the objective of the previous solution, the algorithm will keep the new solution. In the case of a higher objective, the temperature will determine based on the difference between the two costs whether to keep the new solution or discard it. The possibility of accepting worsening solutions allows the algorithm to avoid being stuck in local minima. Over time the temperature will reduce based on the cooling schedule. Practically, the allowed difference in cost for worsening solutions in order to still be accepted will decrease. As the temperature decreases, the focus switches from exploring to exploiting the search space. This will allow the algorithm to eventually converge towards a local or global minimum. The performance of simulated annealing is determined by the choice of initial solution, the objective function, and cooling schedule.

The objective function accounts for the impact of both hard as well as soft constraints. Every constraint is assigned its own penalty function including a constraint weight. The cooling schedule proposed makes use of geometric schedule. That means that the temperature will decrease by a constant factor during every step of the algorithm. The choice for the initial temperature and cool-down factor will determine the share of the search space visited.

While geometric cooling is very simple to implement, it has some shortcomings. Since the temperature is calculated by a deterministic schedule, it mostly depends on the variables chosen by the user. Additionally, this schedule also does not take into account the progress made by the algorithm. Alternatively, more complex adaptive cooling schedules will decrease, or even increase, the temperature based on the rate of acceptance for new solutions. While Aycan and Ayav conclude that this method succeeds in satisfying the hard constraint in order to find a feasible solution, implementing a hybrid version or adding reheating to the cooling schedule might obtain higher quality results.

A more complex cooling schedule can be seen in the Simulated Annealing with Reheating (SAR) algorithm by Leng Goh et al. [1]. They propose a two stage hybrid timetabling algorithm where the first stage uses Tabu Search to generate a feasible solution. If such a solution is found, the second stage attempts to improve on it by using SAR. Instead of using geometric cooling, reheating or increasing the temperature is possible. This is based on the assumption that whenever the objective is high, the focus should be on exploration and accordingly whenever the objective is low, exploitation is prioritised. Whenever the search is estimated to be stuck in a local optima, the temperature will be reheated until it manages to escape.

This estimate is made by checking the difference between the best and current objective. If the change in objective is under a certain threshold for a pre-determined amount of iterations, the search is considered to be stuck and reheating will occur. This cooling and reheating repeats itself until an optimal solution has been found or a step limit is reached. An additional benefit to reheating compared to a geometric schedule is that no fine-tuning of variables is needed when extending the run time. Figure 1 showcases the effect that enabling reheating has on the temperature, with its value increasing whenever it gets stuck in a local optimum. As a consequence, the search is allowed to explore more, resulting in a higher amount of variation of the objective. In this case, the reheating allowed the search to discover a more optimal solution.



(a) SAR with reheating disabled

(b) SAR with reheating enabled

Figure 1: Effect of reheating on the temperate and objective for Simulated Annealing as discovered by Leng Goh et al. [1]

3.2.3 Adaptive Large Neighbourhood Search

ALNS [30] works by generating new solutions by constantly making changes to the current solution in order to explore the neighbourhood space. These changes are obtained by removing part of the variables and then reintroducing new values in order to generate neighbours. ALNS is considered adaptive since it keeps track of the performance of certain operations and adjusts its parameters accordingly.

Sørensen and Stidsen [31] propose a version of ALNS, building on the more general Large Neighbourhood Search (LNS) algorithm. LNS works by creating new solutions by applying a "destroy" and "repair" operation. Every step a destroy operation will remove a set of variables from the problem, before reintroducing new variables in order to create new solutions. By changing multiple variables, LNS is able to escape local minima. In the proposed ALNS extension however, the single destroy and repair operations are replaced by multiple operations, chosen at random during execution. This changes the original deterministic model into a stochastic version, introducing randomness. Additionally an adaptive layer analyses the impact of each operator and increases the probability of operators having a positive impact on the objective function.

3.2.4 Great Deluge

Dueck [32] first proposed a Great Deluge (GD) algorithm as alternative to Simulated Annealing. It is built on the principle that the search space is limited by an ever changing water level. Newly generated solutions are only considered whenever they are superior compared to the objective threshold level that is represented by the water level. New solutions are found by performing low-level heuristics such as swapping two exams. While Dueck makes use of a simple linear function to determine the water level, different functions can be applied instead. While Simulated Annealing accepts worsening solutions based on a probabilistic value, GD has a hard cut-off point for accepting solutions.

Burke et al. [33] are the first to apply an algorithm based on Great Deluge to the exam timetabling problem. Their reason for using GD is to be able to control the exact run time of the search while still allowing the search to converge. Although this could be done with the Simulated Annealing algorithm for example, determining the exact cooling rate for a correct execution time is often too complex. Instead they propose a GD implementation that they call a degraded ceiling algorithm. Instead of a changing water level, they describe a lowering ceiling with the ceiling representing the upper boundary of the objective function allowed. By slowly moving the ceiling down, the run time before convergence can be accurately determined. This is valuable for university administrators responsible for creating exam timetables since they could let the search run overnight or during the weekend.

By slowly lowering the ceiling, exploration to worsening solutions is feasible as long as the ceiling level has not cut off certain sections of the search space. As the ceiling lowers, the search space becomes tighter and priority shifts to exploitation. This continues until no further improvement is feasible and the stopping criterion is met. They conclude that this degraded ceiling implementation is not only superior to a time limited simulation annealing version, but that running this algorithm for long periods can produce extremely good results. Based on their experiments, degraded ceiling can outperform most algorithms whenever a long run time is feasible.

Kahar and Kendall [34] propose an modified Great Deluge algorithm where the water level or decay rate is dynamically changed. Whenever no improvements are found, the change in water level can be reversed and the decay rate adjusted in order to allow for more exploration. Based on their experiments using data from Universiti Malaysia Pahang, the modified GD algorithm outperforms both the university's proprietary software and the original algorithm by Dueck [32]. Lnasya Syafitrie and Komarudin [35] implemented their own version based on this modified algorithm, with the main change being a slower rate at which the water rises. They find that their version outperforms that of Kahar and Kendall.

McMullan [36] has implemented the Great Deluge algorithm with the addition of a reheating step, similar to the reheating used for Simulated Annealing with Reheating [1]. This better allows the search to escape local optima. While the general GD algorithm would terminate after observing no improvement for an amount of time steps, the proposed version will apply a one-time reheating, reducing the water level again. this allows certain worsening solutions to be accepted again resulting in a larger possible search space. Afterwards the decay rate is increased compared to the initial rate. Experiments using small, medium, and large benchmark data sets show that proposed GD algorithm outperforms other implementations such as Local Search and ant algorithms on the medium data sets. For the small and large data sets, no notable difference is noted.

3.3 Population-based Meta-heuristics

3.3.1 Genetic Algorithms

Genetic Algorithms (GA) [37] are based on the process of natural selection witnessed in nature. It generally consists of several base steps before reaching a final solution. First, the algorithm creates an initial population consisting of feasible solutions. An iterative approach is then taken in order to evolve this population into new generations. For every generation, the fitness of each individual in the population is determined and the 'fittest' individuals are chosen as parents for the new generation. In order to obtain new offspring, genetic operators such as crossover and mutation are applied. Crossover works by combining information of parent solutions into a new offspring. This can be done by swapping exam assignments across the parents. Mutation on the other hand randomly changes assignments in order to create new offspring. This is done in order to maintain randomness, allowing GA to escape local minima. The flow of Genetic Algorithms can be seen in Figure 2.

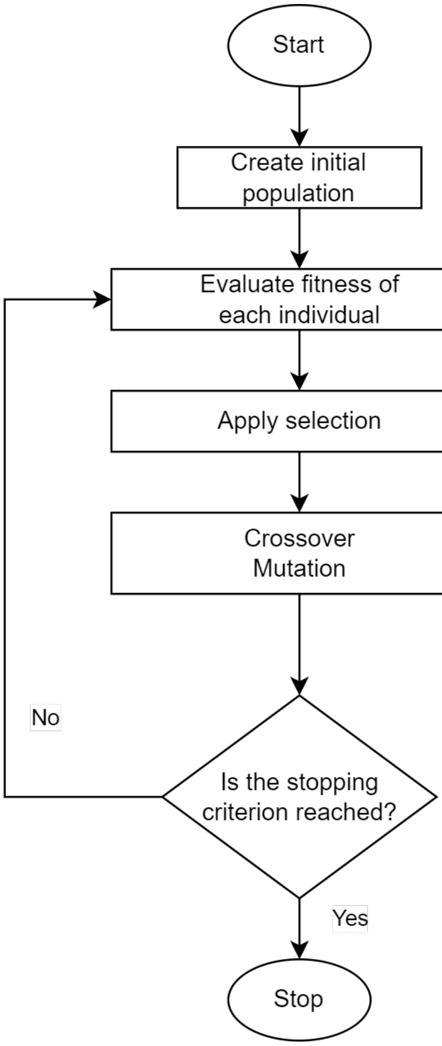

Start
Create initial population
Evaluate fitness of each individual
Apply selection
Crossover
Mutation
No
Is the stopping criterion reached?
Yes
Stop

Figure 2: Genetic Algorithms flow

The paper submitted by Pillay and Banzhaf [38] showcases a two-phased approach to create feasible timetables. Both phases utilise a GA implementation in order to come to a solution, with the first phase focusing on producing a timetable that does not violate any hard constraints. The second phase later attempts to optimise the objective of the soft constraints. Up until 2007, most papers applying GA used data sets specific to a particular institution. This algorithm is tested on the Carter benchmarks [14], allowing its performance to be compared to different approaches.

During the first phase, domain specific heuristics are used instead of random operations in order to create the initial population. While these heuristics require domain knowledge, the quality of the initial population should be superior. In order to determine the next exam to be assigned, the obtained heuristics make use of the number of conflicts, the number of students enrolled per exam, or the number of students with conflicts. These heuristics were then compared to random and best-slot scheduling. It was found that using heuristics succeeded in lowering the soft constraints objective compared to other scheduling methods. For the iterative steps, the fitness function is described as the amount of conflicts per timetable. In order to select the parents with which to generate the new generation, tournament selection is applied. Here, a determined amount of individuals are randomly chosen from the population to be compared against each other. The individual having the lowest objective ends up being selected. This process is repeated until all parents have been selected. When creating new offspring, both mutation and crossover operators were considered. During mutation, a random amount of conflicting examinations are rescheduled. Moreover, crossover operations that randomly swap time slots between two parents were tested. This requires an additional repair mechanism to remove duplicate examinations and schedule missing examinations. However, they note that the crossover operations applied did not positively impact the quality of the solutions. For the final implementation, only the mutation operation on single parents is used to create offspring.

The second phase focuses on minimising the objective of the soft constraints in order the generate a more optimal timetable. While very similar to the first phase, mutation operations are now continuously applied until an offspring superior to the parent is found or until a certain amount of iterations has been reached. The performance of this genetic algorithm was eventually compared to other studies using the same benchmark including Tabu Search and lLarge Neighbourhood Search. While no optimal solutions were found for any of the data sets, equal or improved solutions were obtained compared to alternative methods. They conclude that their implementation equals or even outperforms other evolutionary algorithms on the Carter benchmarks.

3.3.2 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) [39] is derived from the behaviour by collective species as seen in fish schools and bird flocks. A swarm of particles (see fish or bird), representing solutions, moves through the search space at a certain speed and direction.

Chen and Shih [40] discuss a Particle Swarm Optimisation implementation combined with Local Search. During PSO, a swarm of particles, corresponding to different timetables, moves through the available search space. During every iteration, each particle will remember its best encountered position and share its information with the rest of the swarm. The particles will then adjust their velocity and direction on both their personal and global best. The velocity can be seen as the magnitude of the allowed change per time step. Additionally, a local search iteration is applied to explore the surrounding space for a better solution. In order to avoid particles being trapped in a local optimum, a disturbance operation is introduced [41]. This operation forces particles to move towards unexpected directions, that aren't based on previous experience. Doing this, the amount of exploration is increased.

Tassopoulos and Beligiannis [42] builds on the general Particle Swarm Optimisation algorithm. Initially, they start with a high amount of 150 particles. Whenever the fitness value of a produced particle exceeds a certain tolerance value, this particle is set to inactive and will not be used in further steps. This tolerance value is calculated during each step in order to deactivate weak particles at the start but make it harder for particles to reach that threshold down the line. When the amount of particles is reduced to 30, this procedure stops. The reasoning behind starting with a large amount of particles is to make it possible to explore a wider search space while keeping the execution time down by reducing the particles over time. Similar to Chen and Shih's approach [40] a Local Search algorithm is used to minimise one of the soft constraints. Results show that it outperforms the Genetic Algorithms (GA) implementation that their PSO version was tested against.

3.3.3 Honey-Bee Mating Optimisation

Honey-Bee Mating Optimisation (HBMO) [43] are based on the mating of honey-bees as witnessed in nature. During the mating procedure, the queen representing the current best solution is able to mate with other bees in order to produce new solutions. Since specific terms used in the natural mating process are used when describing the optimisation algorithm, Table 1 provides translation of the analogy. HBMO follows the natural process when the queen leaves the hive on a mating flight. Here she maintains a certain speed and direction, creating the possibility of drones to mate with her. After mating with a drone, its genetic information is stored within the queen to be used in the breeding phase. After each mating, the queen will change her energy and speed. As soon as the queen reaches a certain energy threshold or reaches a mating limit, she will return to the hive. Upon return, the queen will randomly select genetic information obtained and perform a crossover step to create a new brood. Every brood is fed by a worker in order to improve the obtained solution. After every brood is improved, the fittest brood is compared to the queen. If the brood corresponds to a superior solution, the queen is replaced by the brood. Both the queen and the other broods are killed and a new mating flight will commence.

Sabar et al. [44] are the first to propose the use of Honey-Bee Mating Optimisation in order to solve the exam timetabling problem. The proposed variant on the original algorithm [43] attempts to solve HBMO's weakness of suffering from early convergence. Originally, the drones, that are used to mate with the queen, are never replaced which reduces the amount of variation. This is solved by replacing the drones that were successful in mating with the newly created broods. Additionally, after crossover, the heuristic applied by the worker is based on Local Search in order to optimise the brood as much as possible.

While HBMO is very similar to other population based methods such as genetic algorithms, two clear differences can be noted. First, HBMO maintains the queen as one of the two ’parents’ used during crossover. Since the queen is considered as the current best solution, this is meant to improve obtained solutions. Secondly, the Local Search applied by the worker can be considered an exploitation phase which is not present in Genetic Algorithms. Finally, Sabar et al. conclude that the proposed HBMO alternative manages to create comparable or superior solutions compared against other population-based methods.

Table 1: Analogy between the natural mating process and the HBMO algorithm

Natural honey bee	Artificial honey bee
Queen	Current best solution
Drones	Possible solutions
Broods	Newly generated solutions
Worker	Heuristic search
Mating or Breeding	Crossover

3.4 Exact Methods

3.4.1 Integer Programming

Kristiansen et al. [45] describe a Mixed-Integer Programming (MIP) model designed to solve XHSTT timetabling data sets. The XHSTT format was formulated to standardise timetabling data sets in order to compare heuristics. The proposed model makes use of two stages. During the first stage, a simplified MIP model is generated taking only the hard constraints of the problem into account. This stage is ran until a specified amount of time has passed or until the model has been solved to optimality. The benefit of using integer programming here is that MIP models can issue ’certificates of optimality, indicating that an optimal solution has been reached. This differs from heuristic methods where a model cannot guarantee that an optimal solution has been found unless the objective function is brought to 0. With MIP models, one can determine whether the cost generated by the hard constraints is the most optimal solution feasible. This allows a clear cut off point for the model to stop focusing on the hard constraints solely.

If a certificate of optimality can be generated, the second stage is executed. Otherwise, the algorithm ends. Before solving the model in stage 2, the soft constraints are added again. Additionally, an extra constraint is added which keeps the optimal value of the cost generated by the hard constraints. The second stage ends after the remaining time after stage 1 has passed. The proposed model was not only successful in creating 2 new solutions to XHSTT data sets, it was also able to prove optimality of previously found solutions. Additionally, it could also provide the lower bounds for several other data sets and improve the best solution found so far. These lower bounds are crucial in order to compare the quality of solutions that have not reached optimality.

Al-hawari et al. [46] attempts to solve the university exam timetabling problem by splitting the problem into three smaller sub-problems in order to significantly decrease the amount of variables required in the formulas used and thus reduce the processing power and storage capacity needed. Additionally, the simplification of the formulas also improves the explainability of the model, making it easier to understand. The initial problem is split into 3 problems, each sub-problem continuing on the solution provided by the previous phase. Phase one and two will use a graph colouring Integer Programming (IP) model to generate a feasible solution. Graph colouring works by assigning colours to the vertices while avoiding that two vertices connected by an edge are assigned the same colour. These vertices are called adjacent. Additionally, graph colouring will attempt to use the least amount of colours possible to generate a feasible solutions.

The first phase starts by assigning time slots to the different exams. Different from stated in the problem definition, Al-hawari et al. consider a time slot to be a moment in time when exams are being held. For example, during an exam period of 20 days with two exam moments a day, there would be 40 time slots available for exams to be planned. Initially, these time slots are solely an exam moment and do not include a day yet. During the first phase, exams are seen as the vertices with two vertices being connected by an edge if a student is enrolled in both exams. The time slots act as colours and are assigned to the vertices. Every feasible solution produced by this phase will satisfy the hard constraint that a student can not have more than one exam at the same time. Phase two builds upon this solution by assigning days to the time slots, meaning that the time slots are now considered vertices and the different days colours. Again, vertices are adjacent if the exams assigned to the time slots share mutual students. The new solution will have added the extra constraint that a student can’t have two exams on the same day. Lastly, the third stage will assign all exams to rooms, keeping into account the exam enrolment and room capacity. The final solution, if feasible, will satisfy all hard constraints. An important remark to note is that no soft constraints were introduced for creating an improved exam distribution. As a result, a feasible solution will only make sure that a student has no exams on two following days or on the same day.

3.5 Hyper-heuristics

While many optimisation methods have proven successful in generating optimal solutions, many of these methods are designed for a single problem and require domain-specific knowledge to be efficient. Hyper-heuristics attempt to solve this problem by providing a problem-independent solution which allows different methods to be used across problems. They can be considered either selective or generative. Selective hyper-heuristics choose from a predefined set of heuristics depending on the problem encountered, while generative hyper-heuristics procure new heuristics by combining multiple heuristics.

Kheiri and Keedwell [47] focus on the selection hyper-heuristics to solve the high school timetabling problem. Every iteration in the solution process, consists of two steps. First, low-level heuristics are applied. These heuristic apply basic operations to create new solutions. Secondly, a move acceptance method will determine whether the keep or discard the new solution. In their study, a set of 15 low-level heuristics are employed including randomly rescheduling or unscheduling an exam, shuffling the time slots of several exams, changing the room of an exam, etc. For the move acceptance method, several methods such as Hill Climbing, Simulated Annealing and, Great Deluge are compared. The choice of acceptance method will determine when a new solution is accepted. In hill climbing, a solution will only be accepted if its better compared to the previous solution. On the other hand, in Simulated Annealing, worse solutions can be considered due to the probability determined by the temperature.

The hyper-heuristic proposed by Kheiri and Keedwell proposes a sequence-based selection where each time a sequence of heuristics is applied instead of a single heuristic. Experiment results show that these sequences outperform single applications of heuristics and that the selection of the low-level heuristics is more important than the move acceptance method used. They finally conclude that the proposed method provides solutions superior to those of previous popular methods.

Kendall and Mohd Hussin [48] describe a hyper-heuristic where the order of the low-level heuristics applied to the current solution is determined by a Tabu Search implementation. At each step of the search, the heuristic with the best performance, if not tabu, is chosen to explore its own search space. The heuristics chosen then become tabu, which means that they will not be considered in the next iterations. Because of this, the search space of other low-level heuristics that might perform worse will also be explored. The size of the tabu list becomes a variable to be optimised for each problem instance. The same set of 13 low-level heuristics proposed during one of their previous experiments was reused [49]. Kendall and Mohd Hussin conclude that the hyper-heuristic is able to find solutions that are 80% better compared to manual generated solutions based on their objective function.

4 Method

For this thesis, the Tabu Search algorithm, as proposed by Alvarez-Valdes et al. [24], is used to conduct our experiments. TS was chosen for several reasons. First, the simple nature of TS improves the interpretability and explainability of the method. The interpretability is important when trying to improve an algorithm based on its results. Only after understanding why a search method behaves a certain way can it efficiently be improved. High explainability makes it easier to explain the used algorithm to university administrators. Secondly, the tabu list prevents cycling between a set of states, which can reduce the amount of time needed to converge. Finally, it has been shown that Tabu Search is able to outperform other competitive algorithms on data sets specific to institutions [24] [27] [26] as well as standardised benchmarks [25].

4.1 Tabu Search

Tabu Search (TS) [22] starts by constructing an initial solution. The solution can be generated randomly or by applying a deterministic approach. During the entire process, both the best seen solution to date and the current solution are maintained. Keeping the best seen solution is necessary in order to allow the current solution to accept allowing worsening changes. This helps TS to avoid getting trapped in local minima. After the solution initialisation, the iterative procedure starts searching for a feasible solution. This loop ends when a specified stopping condition is met. This condition is generally a combination of a solution its objective function scoring below a certain threshold or the procedure reaching a maximum number of iterations.

The first step of the main loop generates the complete list of possible neighbours for the current solution and ranks them based on the objective function. Subsequently, the best neighbour, for which the required move is either not tabu or that meets the aspiration criterion, is chosen as the next solution. A move is considered tabu if it is present within the tabu list, meaning it has been used recently. The aspiration criterion is added to be able to override the tabu requirement. A possible aspiration is to accept solutions that are better than the best seen solution. After choosing the next solution, the best seen solution is updated if the newly generated solution is superior based on the objective function.

When the stopping criteria is met, the algorithm ends and the best solution is returned. The full flow of the algorithm can be seen in figure 3.

4.2 Implementation frameworks

The algorithm implemented to run the experiments is written using Python3.10. The code requires no packages that are not in the Python Standard Library. The profiler cProfiler is used to analyse the performance. It generates a list detailing how long each part of the algorithm runs for and allows to pinpoint and improve bottlenecks.

4.3 Implementation: version 1

The proposed algorithm makes use of a 2-phased approach. During phase 1, the emphasis is on the hard constraints. More specifically, reducing the amount of common students conflicts on the same day is prioritised. Optimally, this initialisation phase would return a solution that is already feasible. The second phase can be considered an optimisation phase, which tries to satisfy the soft constraints as much as possible and thus attempt to distribute the exams evenly.

4.3.1 Initialisation phase

The algorithm starts with the initialisation for which the pseudo code is shown in Algorithm 1. First, the initial solution must be generated. In the proposed algorithm this is done by randomly assigning all exams to time slots. Generally, this will result in several hard constraint violations, such as students having more than 1 exam on the same day and room capacity being exceeded. Since Alvarez-Valdes et al. consider the room capacity a soft constraint, this is acceptable and will be improved during the iterations. However, in this case, room capacity is seen as a hard constraint that can't be violated at all. In order to circumvent these capacity violations, exams are sorted by student count and only then randomly assigned to time slots with rooms having sufficient capacity. As long as the time slot quantity and room capacity is sufficient, this will result in a solution with no room capacity violations. Another added complexity is the presence of different room types which was not the case for Alvarez-Valdes et al. In order to satisfy constraint 2, only rooms with a suitable type are considered when assigning exams to time slots.

After generating the initial solution, the iterative procedure starts by generating the set of neighbours of the current solutions. In the search space X, we consider a solution $s' \in X$ a neighbour of $s \in X$, whenever we can move an exam to a time slot in a different period. Evaluating the entire search space would be too time consuming. Instead, we

Algorithm 1: Initialisation phase

```
Generate an initial solution s in the space of solutions X;
$s^* = s$ (with $s^*$ the best solution seen so far);
$k = 0$ (with k the number of iterations);
while k < maximum number of iterations and $f(s^*) \neq 0$ do
    $k = k + 1$;
    /* perform single moves of one exam to another time slot */
    Generate set of solutions $V^* \subseteq N(s, k)$ (set of neighbours of s);
    Sort $V^*$ by ascending $f(s')$ and select the best $s'$;
    foreach solution $s'$ in $V^*$ do
        if not $tabu(s')$ or $f(s') < f(s^*)$ (aspiration criterion) then
            $s = s'$;
            break;
        end
    end
    if $f(s') < f(s^*)$ then
        $s^* = s'$;
    end
end
return $s^*$
```

first sort all periods by its contribution to the objective function. From the most conflictive period, we select the most conflictive time slot. Finally, we calculate all available time slots that we can swap with. In order to swap two time slots, the two affected exams (or sole affected exam when swapping to a time slot with no scheduled exam) must be able to be scheduled in the new time slots keeping the room type and capacity into account. This will ensure that no additional constraint 2 and 3 violations are introduced.

For each possible move, the objective score is calculated in order to rank all neighbours. The objective function for a solution s is defined as

$$f(s) = \sum_{i=1}^{P} \left(\sum_{j \in E_i} \sum_{\substack{k \in E_i \\ k \neq j}} w|S_j \cap S_k| \right) \tag{1}$$

with P the amount of periods, E_i the set of exams scheduled to period i, S_j and S_k the students scheduled to exam j and exam k respectively. This makes $|S_j \cap S_k|$ the amount of common students between the two exams. Lastly, the weight of student conflicts is denoted by w.

After sorting all found neighbours by objective function, the best solution, that is not tabu or for which the aspiration criterion applies, is chosen. The aspiration criterion accept a solution, for which the required move is tabu, as long as it is superior to s^* (the best solution seen so far) If no moves are possible, we select the next most conflictive time slot until a suitable move has been found. When choosing the new solution, the tabu list is updated with the new tabu move consisting of the exam and period involved. Whenever the tabu list exceeds its maximum size, the oldest move is deleted, in order to allow that move again in future iterations.

This phase runs until a solution without conflicts has been found or the maximum number of iterations has been reached. For the latter case, the schedule will have conflicting exams during the same period with common students.

4.3.2 Optimisation phase

After the initialisation phase, we start the optimisation of the solution. Here the focus is including the soft constraints in the objective function to generate a solution that is the most optimal. The foundation of this phase is similar compared to the first phase with some distinct features. The overall flow can be seen in the pseudo code described in Algorithm 2.

First, the objective function is updated to take the distribution of exams into account. The objective of a solution is now defined as

Algorithm 2: Optimisation phase

```
Data: Solution s (the result of the initialisation phase)
$s^* = s$ (with $s^*$ the best solution seen so far);
$k = 0$ (with k the number of iterations);
while $k$ < maximum number of iterations and $f(s^*) \neq 0$ do
    $k = k + 1$;
    if $k \bmod 3 \neq 0$ then
        /* perform single moves of one exam to another time slot        */
        Generate set of solutions $V^* \subseteq N(s, k)$ (set of neighbours of s after single move);
        Sort $V^*$ by ascending $f(s')$ and select the best $s'$;
        foreach solution $s'$ in $V^*$ do
            if not $tabu(s')$ or $f(s') < f(s^*)$ (aspiration criterion) then
                $s = s'$;
                break;
            end
        end
    else
        /* swap entire period                                           */
        Generate set of solutions $V^* \subseteq N(s, k)$ (set of neighbours of s after swapping periods);
        Sort $V^*$ by ascending $f(s')$ and select the best $s'$;
        $s = s'$;
    end
    if $f(s') < f(s^*)$ then
        $s^* = s'$;
    end
end
return $s^*$
```

$$f(s) = \sum_{i=1}^{P} \sum_{j=i}^{P} \left(w_{|i-j|} \sum_{k \in E_i} \sum_{\substack{l \in E_j \\ k \neq l}} |S_k \cap S_l| \right) \tag{2}$$

with P the amount of periods, E_i the set of exams scheduled to period i, S_k and S_l the students scheduled to exam k and exam l respectively. This makes $|S_k \cap S_l|$ the amount of common students between the two exams. Here, the weight w is dependent on the amount of days between the exams, with w_i the weight for exams scheduled i days apart.

While the objective function in the first phase only used the common students between exams during the same period, the new objective function will take all periods into account. Constraint 5a, that does not allow students having more than one exam on the same day, is now relaxed to constraint 5b. By having a large weight w_0 for two exams scheduled on the same day, these exam conflicts for non-model track are only introduced if the decrease of the objective function is significant. Additionally, placing exams at a further distance will now be preferred since the weight w_{x+1} will be smaller than the weight w_x.

Secondly, two options are now available when generating the set of neighbours of the solution. The first option is a copy of the process in the initialisation phase where the set of neighbours of the solution is generated by performing single moves. Then the best solution that is either not tabu or that meets the aspiration criterion will be used. While this option might introduce constraint violations, the high w_0 will discourage and, depending on the values used, prevent this. This method will be alternated with a period permutation. Instead of moving a single exam, two entire periods will be swapped. Since the exams in the periods are not updated, this swap can not introduce any hard constraint violations. By moving entire periods, the objective function can change significantly in a single move. However, the amount of available moves is much lower and can be defined as

$$\begin{aligned} \text{\# possible moves} &= \binom{P}{2} \\ &= \frac{P!}{2!(P-2)!} \end{aligned} \tag{3}$$

4.4 Implementation: version 2

After running experiments using version 1, some run time and result characteristics were observed. Version 2 diverts even more from the original algorithm in order to avoid these occurrences. First, the objective function originally defined in the initialisation phase as

$$f(s) = \sum_{i=1}^{P} \left(\sum_{j \in E_i} \sum_{\substack{k \in E_i \\ k \neq j}} w|S_j \cap S_k| \right) \tag{4}$$

has an unwanted characteristic: it does not punish students with several exams on the same day extra. For exam schedules without a conflict free solution, this often results in students, who are enrolled in smaller exams, having a large number of exams on the same day. In order to avoid this behaviour, the objective function is updated to

$$f(s) = \sum_{i=1}^{P} \left(\sum_{s \in S(E_i)} w^{c_{s,i}-1} \right) \tag{5}$$

with P the amount of periods, E_i the set of exams scheduled to period i, $S(E_i)$ the set of students with more than 1 exam during E_i, $c_{s,i}$ the number of exams student s has on period i, and w the weight of conflicts. By placing $c_{s,i}$ in the exponent, having multiple exams on the same day is punished severely.

Secondly, the way a tabu move is defined was changed. Originally, when moving an exam to a different period, the combination of the exam and period was used to define the tabu move. This often resulted in the scenario that the same exam was continuously being moved. Since this generally happened to one of the largest exams, it always resulted in a large impact on the objective function and kept being presented as the most conflictive time slot. This behaviour can be avoided by having the exam as the sole identifier of the tabu move. By doing this, a single exam will not be moved as frequently.

4.5 Implementation: version 3

Version 1 and 2 both reduce the search space by only looking at the most conflictive time slot of the most conflictive period in order to determine the possible moves. Different time slots are only looked at when no moves are possible. This version steps away from that principle and continues evaluating the next most conflictive time slots their possible moves until a specified amount of possible moves have been found. Only then it selects the move generating the best next solution.

5 Experimental Design

For this case study, the data set to be used is provided by the FTI and FWET faculties of the University of Antwerp (UA). The data consists of the exam information for the January and June exam period of 2021 and the schedule used. Even though the two faculties compose their own schedules, the problems are not independent. A section of the rooms are available for both faculties. As a consequence, the schedule has to be generated together and only then be split up. Otherwise a shared exam room could be double booked. The statistics for the data sets provided can be seen in Table 2 and 3. The fact that the scheduling problem for FTI and FWET is not independent can be seen here as well. The statistics for FTI and FWET combined do not equal the sum for the statistics of the faculties separated. The amount of time slots can be calculated as follows:

$$\text{\# of time slots} = \text{\# Rooms} \times \text{\# Periods} \times \text{\# Exam times} \tag{6}$$

with there being two exam times, namely a morning and afternoon exam. For the January and June schedule there are respectively 20 and 24 periods.

Table 2: Data set statistics for January 2021

	FTI	**FWET**	**FTI+FWET**
Students	770	1041	1811
Exams	185	360	536
Rooms	15	45	54
Time slots	600	1800	2160

Table 3: Data set statistics for June 2021

	FTI	**FWET**	**FTI+FWET**
Students	782	1071	1852
Exams	169	342	491
Rooms	15	45	54
Time slots	720	2160	2592

The hyperparameters available for the different versions can be seen in Table 4.With MAX_ITER_OPTIMISATION and MAX_ITER_INITIALISATION defining the duration of the algorithm run time, the main variables to set are the weights. Unlike P_INITIALISATION being a single value, P_OPTIMISATION is a list with the value at index i containing the penalty for two exams at distance i. If there is no value at index i, the penalty is equal to 0.

Table 4: Tabu search hyperparameters

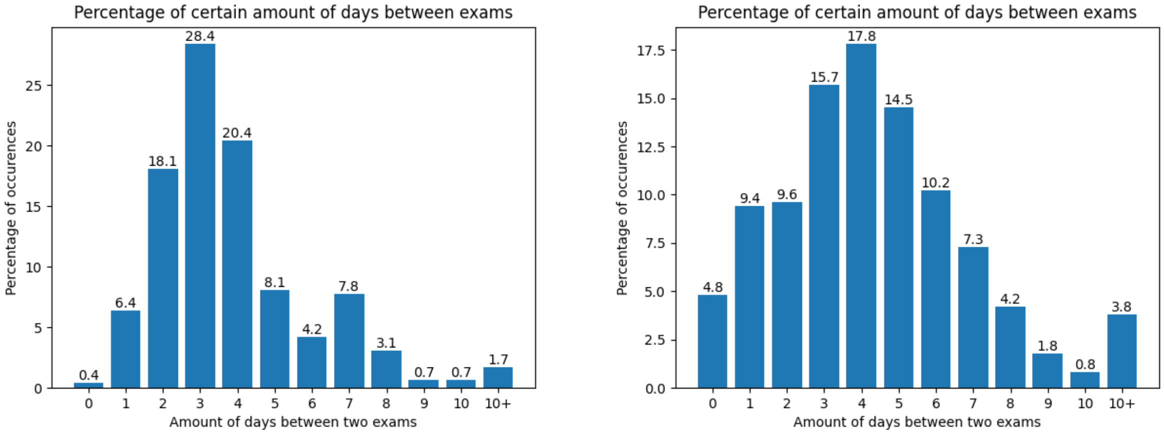
	Parameter	**Description**
All versions		
	P_INITIALISATION	weight of conflicts during initialisation phase
	P_OPTIMISATION	weight of distance between exams
	MAX_ITER_INITIALISATION	max amount of iterations during initialisation phase
	MAX_ITER_OPTIMISATION	max amount of iterations during optimisation phase
Version 3		
	MAX_MOVES	amount of moves to evaluate each iterations

6 Results / Evaluation

In order to evaluate the results from the experiment, we have to verify whether the phases perform their objective as described. Firstly, does the initialisation phase succeed in generating a ‘feasible´ solution namely one without or with a minimal number of hard constraint violations? Secondly, does the optimisation phase succeed in transforming the initial solution into one with an acceptable solution?

6.1 Reference solutions

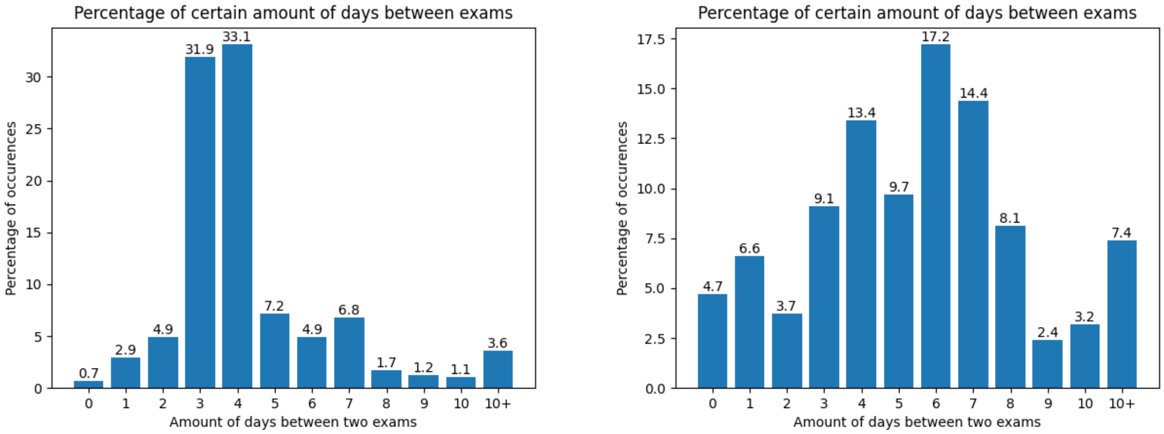
The actual exam schedules, manually created by the administration of the University of Antwerp, can be used as a baseline to compare the generated schedules. These exam distributions can be seen in Figures 4 and 5. Most notably, the manual schedules have a focus on keeping the amount of same day exams to a minimum and having 2 or more days in between exams as much as possible. This distribution is especially visible in the schedules for FTI. Additionally, it can be seen that the schedule for FWET contains a high amount of exam conflicts. In order to be considered a superior solution, automatically generated schedules will have to minimise the amount of exams with fewer than 2 days in between, with a focus on same day exam violations.



(a) FTI

(b) FWET

Figure 4: Manual exam distribution for January 2021



(a) FTI

(b) FWET

Figure 5: Manual exam distribution for June 2021

6.2 Initialisation phase

In order to quantify the feasibility of a solution, it is possible to look at either the objective function or the amount of hard constraint violations. Since the objective function was adapted to contain exponents and the conflict weight having a large impact on the size of the objective, looking at the amount of hard constraint violations can provide a better perspective. By plotting the violations per iterations, the progress made by the initialisation phase can be visualised. Figure 6 shows that the initialisation phase is able to generate a solution for FTI+FWET without hard constraint violations. It reaches a feasible solution after fewer than 150 iterations, generally taking under 500 seconds.

However, Figure 7 shows that no attention was given to the distribution which can be confirmed by the presence of a high percentage of exams after a single day. These figures are generated by calculating the distance between each exam for every student, with the exams being sorted on exam date. For example, a student A with exams on day 1, 5, and 7 will contribute the distance between day 1 and 5, as well as the distance between day 5 and 7. This results in the student contributing to the $x = 4$ and $x = 2$ bar, respectively. We can also look at the case for a student B with conflicting exams. This can be showcased using an identical schedule compared to student A, with the addition of a conflicting exam on day 5. This results in the distances being calculated using the sequence of 1, 5, 5, and 7. Because of this, student B will contribute to $x = 4$, $x = 0$, and $x = 2$ bar, respectively.

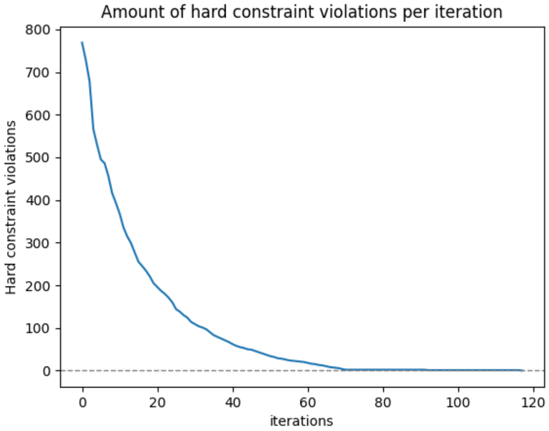
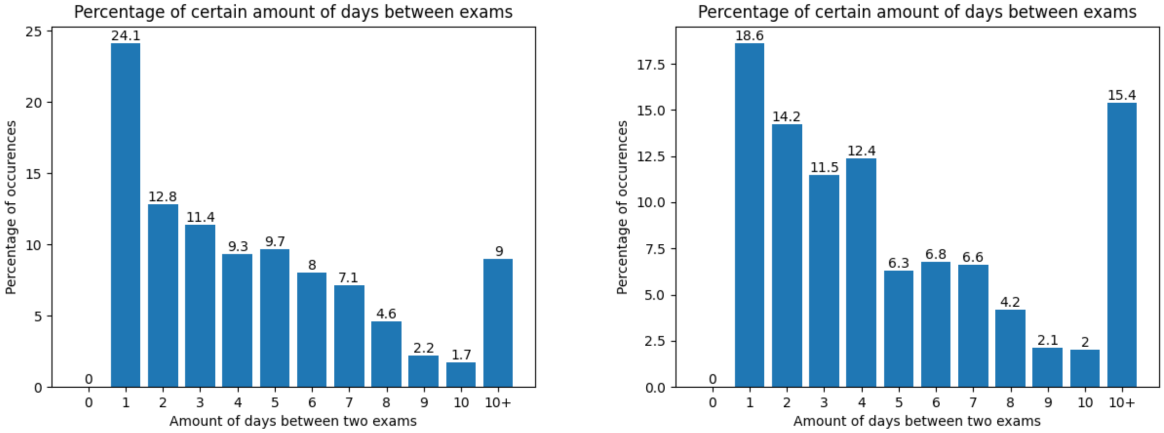


Figure 6: Hard constraint violations



(a) FTI

(b) FWET

Figure 7: Exam distribution after initialisation phase for June 2021

6.3 Optimisation phase

While we have shown that tabu search can efficiently generate a feasible solution from the provided data set, the exam distribution will determine whether the automated schedules are superior in quality compared to the

7 Threats to Validity

8 Conclusions

Acronyms

ALNS Adaptive Large Neighbourhood Search. 14, 15

CB-CTT Curriculum-based Course Timetabling. 13
CTT University Course Timetabling. 10

ETT University Examination Timetabling. 10

FTI Faculty of Applied Engineering. 10, 24–26
FWET Faculty of Science. 10, 24–26

GA Genetic Algorithms. 4, 13, 15–18
GD Great Deluge. 15, 19

HBMO Honey-Bee Mating Optimisation. 5, 17, 18
HTT High-School Timetabling. 10

IP Integer Programming. 18
ITC International Timetabling Competition. 12

LNS Large Neighbourhood Search. 15, 17
LS Local Search. 12, 13, 15, 17, 18

MIP Mixed-Integer Programming. 18

PATAT International Conference on the Practice and Theory of Automated Timetabling. 12
PE-CTT Post-Enrolment Course Timetabling. 13
PSO Particle Swarm Optimisation. 17

SA Simulated Annealing. 4, 13–15, 19
SAR Simulated Annealing with Reheating. 14, 15

TS Tabu Search. 4, 13, 14, 17, 19, 20, 34

UA University of Antwerp. 10, 24, 25

References

[1] Say Leng Goh, Graham Kendall, and Nasser R. Sabar. Improved local search approaches to solve the post enrolment course timetabling problem. *European Journal of Operational Research*, 261(1):17–29, 2017. pages 4, 14, 15

[2] Copyright Stichting, Mathematisch Centrum, and Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13, 01 1996. pages 10

[3] Jeffrey Kingston. Educational timetabling. *Studies in Computational Intelligence*, 505:91–108, 01 2013. pages 10

[4] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5:691–703, 12 1976. pages 10

[5] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. pages 10

[6] Henri Farreny and Henri Prade. Heuristics—intelligent search strategies for computer problem solving, by judea pearl. (reading, ma: Addison-wesley, 1984). *International Journal of Intelligent Systems*, 1(1):69–70, 1986. pages 10

[7] Shen Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21:498–516, 1973. pages 12, 13

[8] Rhydian Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30:167–190, 01 2008. pages 12

[9] Meysam Kohshori and Mohammad Saniee Abadeh. Hybrid genetic algorithms for university course timetabling. *International Journal of Computer Science Issues*, 9, 03 2012. pages 12

[10] Peter Cowling, Graham Kendall, and E Soubeiga. A hyperheuristic approach to scheduling a sales summit. pages 176–190, 01 2001. pages 12

[11] Edmund Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64:1695–1724, 07 2013. pages 12

[12] Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. Educational timetabling: Problems, benchmarks, and state-of-the-art results. *European Journal of Operational Research*, 308, 07 2022. pages 12

[13] Cumming A and Paechter B. Standard formats for timetabling data, unpublished discussion session at the first international conference on the practice and theory of automated timetabling, edinburgh. 1995. pages 12

[14] Michael W. Carter, Gilbert Laporte, and Sau Yan Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47(3):373–383, 1996. pages 12, 16

[15] Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. *Computers Operations Research*, 132:105300, 2021. pages 12

[16] Barry Mccollum, Paul Mcmullan, Edmund Burke, Andrew Parkes, and Rong Qu. The second international timetabling competition: Examination timetabling track. 10 2007. pages 13

[17] Andrew Parkes and Ender Özcan. Properties of yeditepe examination timetabling benchmark instances. pages 531–534, 01 2010. pages 13

[18] Michele Battistutta, Andrea Schaerf, and Tommaso Urli. Feature-based tuning of single-stage simulated annealing for examination timetabling. *Annals of Operations Research*, 252, 05 2017. pages 13

[19] Gerhard Post, Samad Ahmadi, Sophia Daskalaki, Jeffrey Kingston, Jari Kyngas, Kimmo Nurmi, G Post, S Ahmadi, Stavroula Daskalaki, J Kyngas, · Nurmi, and D Ranson. An xml format for benchmarks in high school timetabling. *Annals of Operations Research*, 194:385–397, 04 2012. pages 13

[20] Luca Di Gaspero, Barry Mccollum, and Andrea Schaerf. The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). 01 2007. pages 13

[21] Rhydian Lewis, Ben Paechter, and Barry Mccollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. *Cardiff University, Cardiff Business School, Accounting and Finance Section, Cardiff Accounting and Finance Working Papers*, 01 2007. pages 13

[22] Fred Glover and Eric Taillard. A users guide to tabu search. *Ann. Oper. Res.*, 41:1–28, 03 1993. pages 13, 20

[23] Fred Glover and Manuel Laguna. *Tabu search I*, volume 1. 01 1999. pages 13

[24] Ramón Álvarez Valdés-Olaguíbel, Crespo E., and Tamarit Goerlich José Manuel. A tabu search algorithm to schedule university examinations. *Qüestiió: quaderns d'estadística i investigació operativa*, 21(1), 1997. pages 13, 20

[25] Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. pages 104–117, 09 2001. pages 13, 20

[26] Shu-Chuan Chu and H.L. Fang. Genetic algorithms vs. tabu search in timetable scheduling. pages 492 – 495, 01 2000. pages 13, 20

[27] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Metaheuristics for high-school timetabling. *Computational Optimization and Applications*, 9, 04 1999. pages 13, 20

[28] Esra Aycan and Tolga Ayav. Solving the course scheduling problem using simulated annealing. pages 462 – 466, 04 2009. pages 14

[29] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science (New York, N.Y.)*, 220:671–80, 06 1983. pages 14

[30] Stefan Ropke and David Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775, 2006. Feature Cluster: Heuristic and Stochastic Methods in Optimization Feature Cluster: New Opportunities for Operations Research. pages 14

[31] Matias Sørensen and Thomas Riis Stidsen. High school timetabling: Modeling and solving a large number of cases in denmark. In *Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*, pages 359–364, 2012. pages 15

[32] Gunter Dueck. New Optimization Heuristics. The Great Deluge Algorithm and the Record-to-Record Travel. *Journal of Computational Physics*, 104(1):86–92, January 1993. pages 15

[33] E. Burke, Yuri Bykov, J Newall, and Sanja Petrovic. A time-predefined local search approach to exam timetabling problem. *IIE Transactions*, 36:509–528, 06 2004. pages 15

[34] M N Mohmad Kahar and G Kendall. A great deluge algorithm for a real-world examination timetabling problem. *Journal of the Operational Research Society*, 66(1):116–133, 2015. pages 15

[35] Komarudin Lnasya Syafitrie. A great deluge algorithm for university examination timetabling problem. 06 2022. pages 15

[36] Paul McMullan. An extended implementation of the great deluge algorithm for course timetabling. volume 538-545, pages 538–545, 05 2007. pages 15

[37] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992. pages 15

[38] N. Pillay and W. Banzhaf. An informed genetic algorithm for the examination timetabling problem. *Applied Soft Computing*, 10(2):457–467, 2010. pages 16

[39] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995. pages 17

[40] Ruey-Maw Chen and Hsiao-Fang Shih. t. *Algorithms*, 6:227–244, 04 2013. pages 17

[41] Shang-Jeng Tsai, Tsung-Ying Sun, Chan-Cheng Liu, Sheng-Ta Hsieh, Wun-Ci Wu, and Shih-Yuan Chiu. An improved multi-objective particle swarm optimizer for multi-objective problems. *Expert Systems with Applications*, 37:5872–5886, 08 2010. pages 17

[42] Ioannis X. Tassopoulos and Grigorios N. Beligiannis. Solving effectively the school timetabling problem using particle swarm optimization. *Expert Systems with Applications*, 39(5):6029–6040, 2012. pages 17

[43] Hussein Abbass. Mbo: marriage in honey bees optimization-a haplometrosis polygynous swarming approach. volume 1, pages 207 – 214 vol. 1, 02 2001. pages 17

[44] Nasser Sabar, Masri Ayob, and Graham Kendall. Solving examination timetabling problems using honey-bee mating optimization (etp-hbmo). *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2009)*, pages 399–408, 01 2009. pages 17

[45] Simon Kristiansen, Matias Sørensen, and Thomas Stidsen. Integer programming for the generalized high school timetabling problem. *Journal of Scheduling*, 18, 08 2015. pages 18

[46] Feras Al-hawari, Mahmoud Al-Ashi, Fares Abawi, and Sahel Alouneh. A practical three-phase ilp approach for solving the examination timetabling problem. *International Transactions in Operational Research*, 27, 11 2017. pages 18

[47] Ahmed Kheiri and Ed Keedwell. A Hidden Markov Model Approach to the Problem of Heuristic Selection in Hyper-Heuristics with a Case Study in High School Timetabling Problems. *Evolutionary Computation*, 25(3):473–501, 09 2017. pages 19

[48] Graham Kendall and Naimah Mohd Hussin. A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. pages 270–293, 08 2004. pages 19

[49] Graham Kendall and Naimah Mohd Hussin. *An Investigation of a Tabu-Search-Based Hyper-Heuristic for Examination Timetabling*, pages 309–328. 01 2005. pages 19

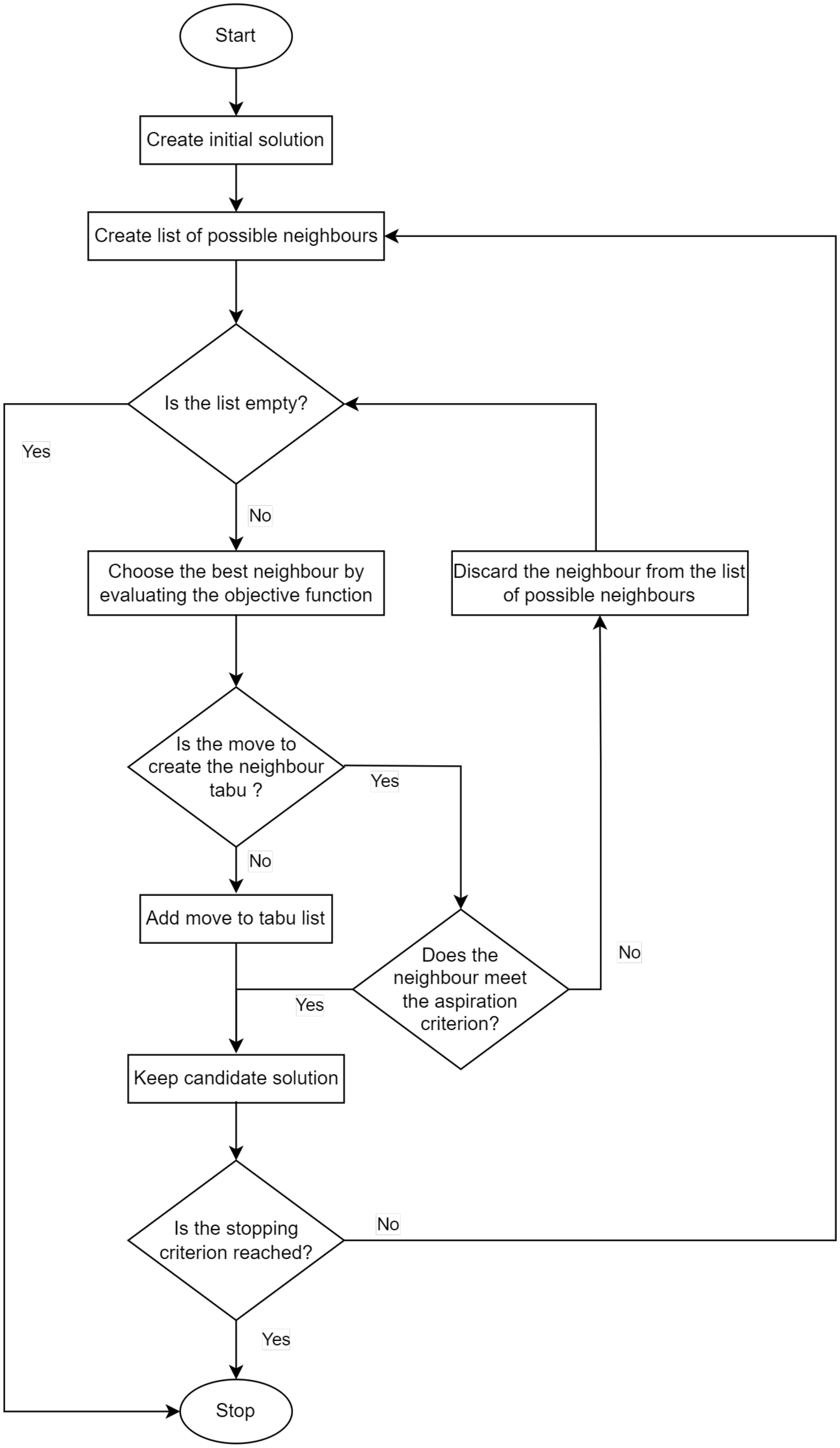

Start
Create initial solution
Create list of possible neighbours
Is the list empty?
Yes
No
Choose the best neighbour by evaluating the objective function
Discard the neighbour from the list of possible neighbours
Is the move to create the neighbour tabu ?
Yes
No
Add move to tabu list
Does the neighbour meet the aspiration criterion?
No
Yes
Keep candidate solution
Is the stopping criterion reached?
No
Yes
Stop

Figure 3: Tabu Search flow