Universiteit
Antwerpen

Artificial Intelligence for optimising the timetables of university exams

De Schepper Cedric

Principal Advisor: Serge Demeyer
Assistant Advisor: Joey De Pauw

May 15, 2023

AnSyMo
Antwerp Systems & Software Modelling
University of Antwerp

Contents

List of Figures

List of Tables

Abstract

English abstract.

Nederlandstalige Samenvatting

Dutch Abstract.

Acknowledgments

Student’s personal acknowledgements.

1 Introduction

We also included a comments command which makes it easier for students and advisors to give feedback or ask questions directly on the text. We can hide all comments in the pdf by changing a single line in the main tex file.

2 Problem definition

2.1 Problem definition

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 constraint. 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 used. After consulting with the responsible parties for the scheduling of the exams for the FTI and FWET faculties, the following constraints were determined:

- Constraint 1: Each exam is scheduled to a timeslot.
- Constraint 2: Each exam is scheduled to a room of the correct type.
- Constraint 3: The number of students enrolled in an exam cannot exceed the capacity of the scheduled room.
- Constraint 4: No student must be scheduled to more than 1 exam on the same day.
- Constraint 5: The number of days between exams must be maximised

Constraint 1 to 4 are considered to be hard constraints with contraint 5 being the sole soft constraint. Formally this means that the first 4 constraints must be satisfied in order to provide a feasible solution, with constraint 5 being taken into account to improve it.

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 which will allow to verify if two exams have common students and would thus conflict if they occur on the same day. Additionally, an exam belongs to a faculty and is of a certain type. Two examples of exam types are oral and written exams.

Rooms The set of rooms present for examinations. Each room has a room type, a student capacity and 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. A time slot is 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 4 ("no student must be scheduled to more than 1 exam on the same day") violations can be determined. 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 will detail different types of algorithms researched to solve the timetabling problem.

Integer programming Integer programming tries to solve an optimisation by minimising or maximising the problem's objective function. Fundamentally, some or all required variables are constrained to integers. Since integer programming makes use of an exhaustive search process, the search space must be reduced as much as possible to reduce the run time.

Simulated Annealing Simulated Annealing [1] makes use of a temperature variable which describes the level of randomness present in the acceptance of a new solution. After generating an initial solution, newly obtained solutions are accepted based on the change in objective function and current temperature. Over time the temperature cools down which reduces the amount of randomness.

Adaptive Large Neighbourhood Search Adaptive Large Neighbourhood Search (ALNS) [2] 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. The use of ALNS for exam timetabling so far has been limited.

Genetic Algorithms Genetic Algorithms (GA) [3] are based on the process of natural selection witnessed in nature. They work by generating an initial set or population of possible solutions. Iteratively, a new population will be formed by selecting the fittest (best) solutions and combining two solutions to create a new generation of solutions.

Particle Swarm Optimisation Particle Swarm Optimisation (PSO) [4] 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.

Honey-Bee Mating Optimisation Honey-Bee Mating Optimisation algorithms (HBMO) [5] 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.

3.1 Integer programming

Kristiansen et al. [6] describes a mixed-integer linear 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. [7] 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 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. Here, 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.

3.2 Simulated Annealing

Aycan and Ayav [8] apply Simulated Annealing in order 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 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.

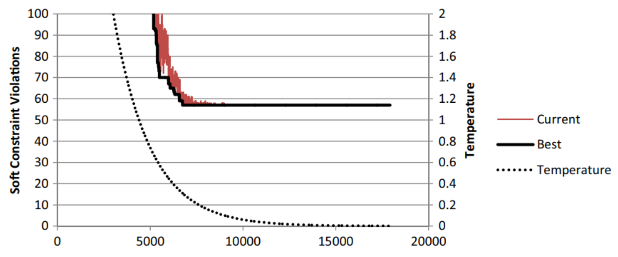
While geometric cooling is a very simple to implement, it comes with some shortcoming. Since the temperature is calculated by a deterministic schedule, it mostly depends on the variables chosen by the user. Additionally, this schedule also does not take into account the progress made by the algorithm. Alternative, the 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. [9] 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.

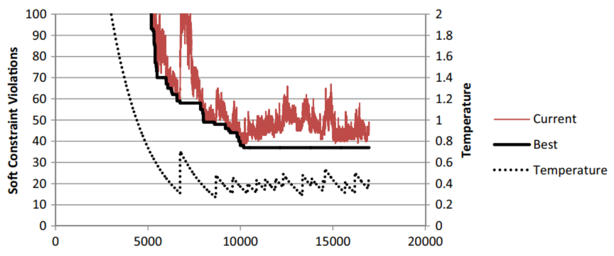
This estimate is made by checking the difference between the best and current objective. If the change in objective is under a certain threshold for a pre-determined amount of iterations, the search is considered to be stuck and reheating will occur. This cooling and reheating repeats itself until an 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 runtime. 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.

3.3 Adaptive Large Neighbourhood Search

Sørensen and Stidsen [10] propose a version of ALNS building on the more general Large Neighbourhood Search 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.



(a) SAR with reheating disabled



(b) SAR with reheating enabled

Figure 1: Effect of reheating on the temperate and objective

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.

3.4 Genetic Algorithms

Genetic Algorithms (GA) are based on the process of natural selection witnessed in nature. It generally consists of several base steps before reaching a final solution. Firstly, 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 iterative flow of genetic algorithms can be seen in figure 2.

The paper submitted by Pillay and Banzhaf [11] showcases a two-phased approach to create feasible timetables. Both phases utilise a genetic algorithm 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 the optimise the objective of the soft constraints. Up until 2007, most papers applying genetic algorithms used data sets specific to a particular institution. This algorithm is tested on the Carter benchmarks, 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 requires 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 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. Since the crossover operations did not have a positive impact on the performance, only mutation was used.

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 large neighbourhood search. While no optimal solutions were found for any of the data sets, equal or improved solutions were obtained compared to alternative methods.

Chu and Fang [12] made a comparison between using genetic algorithms versus Tabu search to obtain time tables. For all of their different experiments, the Tabu search implementation was able to outperform their genetic algorithm both on the quality of the solution as the computational time needed to converge. However, a redeeming quality of Tabu

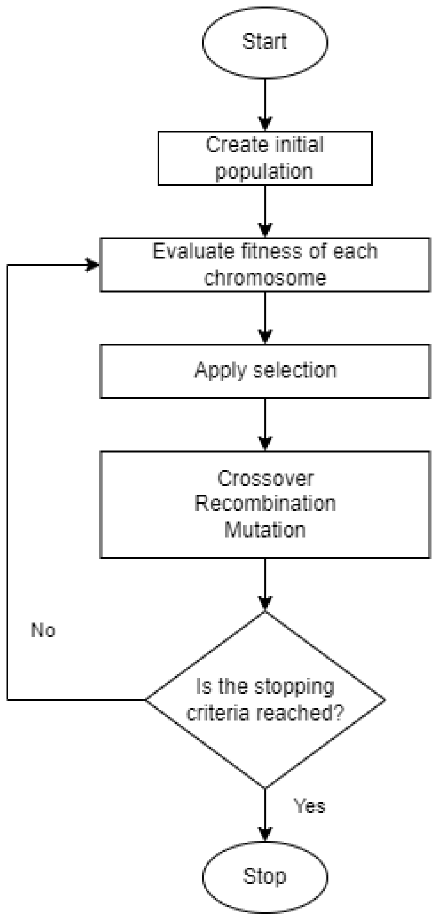

Start
Create initial population
Evaluate fitness of each chromosome
Apply selection
Crossover
Recombination
Mutation
Is the stopping criteria reached?
No
Yes
Stop

Figure 2: Iterative approach by genetic algorithms

search is that it is able to produce several near optimal solutions in one go while Tabu search is limited to a single solution.

3.5 Particle Swarm Optimisation

Chen and Shih [13] 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. In order to avoid particles being trapped in a local optima, an iteration of local search is executed after a particle move. This step will check the surrounding areas for a better solution.

Tassopoulos and Beligiannis [14] 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 [13] a local search algorithm is used to minimise one of the soft constraints. Results show that it outperforms the genetic and evolutionary algorithms that the particle swarm optimisation implementation was tested against.

3.6 Honey-Bee Mating Optimisation

Sabar et al. [15] are the first to propose the use of Honey-bee mating optimisation in order to solve the exam timetabling problem. Since specific terms used in the natural mating process are used when describing the optimisation algorithm, table 1 provides translation of the analogy. The original HBMO algorithm starts 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.

The variant on the original HBMO algorithm proposed by Sabar et al. attempts to solve HBMO's weakness of suffering from early convergence. In the original algorithm, the drones 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. 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, they 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

4 Background

4.1 Tabu Search

5 Method / Experimental Design

5.1 Tabu Search

Tabu search [16] is a meta-heuristic search algorithm based on local search, a heuristic optimisation method that traverses the possible search space by performing local changes to the current solution. Since local search will only accept improving solutions, this traversal can end up being stuck in a local optimum. Tabu search is different from local search by relaxing this rule and also accepting worsening solutions. Additionally, tabu search 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 [17]. 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.

Tabu search starts by constructing an initial solution. The solution can be generated randomly or by applying a deterministic approach. During the entire process, the best seen solution to date is maintained. This is necessary due to tabu search allowing worsening changes or so called moves 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 is to generate the complete list of possible neighbours of the current solution and ranking them based on the objective function. Subsequently, the best neighbour that is either not tabu or that meets the aspiration criterion is chosen as the next solution. A neighbour is considered tabu if it is present within the tabu list. 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.

5.2 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.

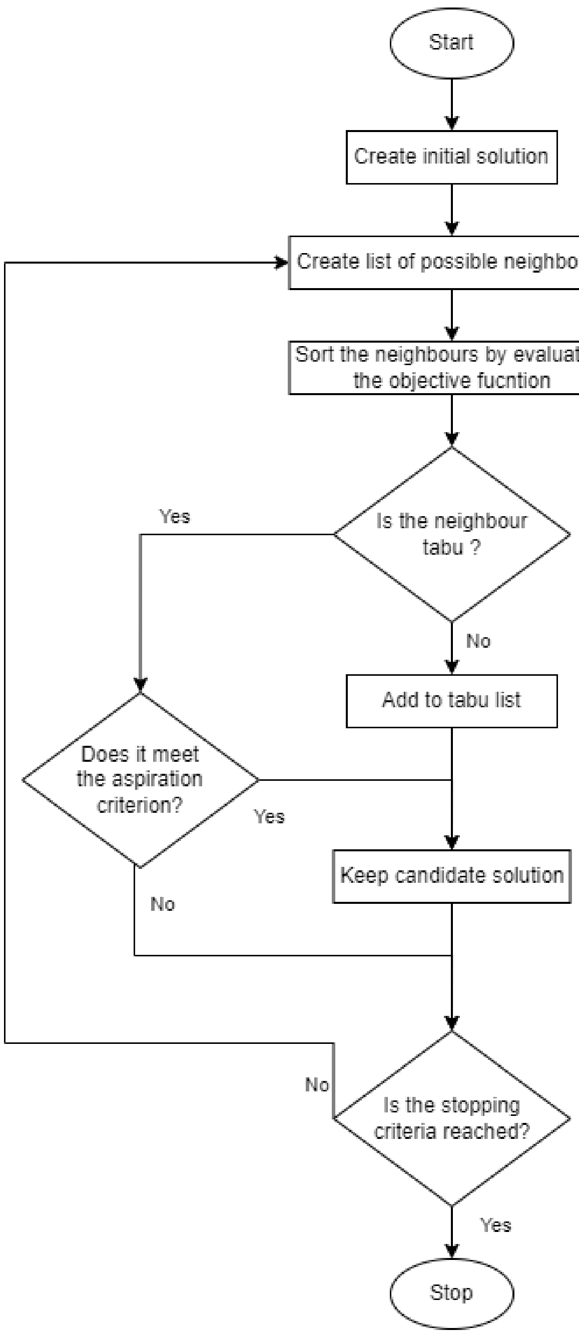
5.3 Implementation: version 1

For this thesis, the initial version or version 1 of the tabu search algorithm to be used in the experiments is based on the algorithm proposed by Alvarez-Valdes et al [18]. The algorithm described 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.

5.3.1 Initialisation phase

The algorithm starts with the initialisation for which the pseudo code is shown in algorithm 1. Firstly, 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 have 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. 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


Start
Create initial solution
Create list of possible neighbours
Sort the neighbours by evaluating the objective fucntion
Is the neighbour tabu ?
Yes
No
Add to tabu list
Does it meet the aspiration criterion?
Yes
No
Keep candidate solution
Is the stopping criteria reached?
No
Yes
Stop
Figure 3: Tabu search flow

different period. Evaluating the entire search space would be too time consuming. Instead, we 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 can be defined as

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 $s^*$
```

$$f(s) = \sum_{i=1}^{N}(\sum_{j,k \in E_i} px_{jk}) \tag{1}$$

with N the amount of periods, E_i the set of exams scheduled to period i, x_{jk} the number of common students between exams j and k, and p the weight of conflicts.

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 accepts solutions that are tabu as long as they are 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.

5.3.2 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.

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

$$f(s) = \sum_{i=1}^{N}\sum_{j=1}^{N}(p_{|i-j|}\sum_{k \in E_i}\sum_{l \in E_j} x_{kl}) \tag{2}$$

with N the amount of periods, E_i the set of exams scheduled to period i, x_{kl} the number of common students between exams k and l, and $p_{|i-j|}$ the penalty for two exams scheduled at a distance of $|i - j|$ periods or days. 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 combination into account. By having a large penalty p_0 for exams scheduled on the same day, the hard constraint of having no exams on the same day for a student is still being prioritised. Additionally, placing exams at a further distance will now be preferred since the penalty p_{x+1} will be smaller than the penalty p_x.

Algorithm 2: Optimisation phase

```
Data: Solution s (the result of the initialisation phase)
1  $s^* = s$ (with $s^*$ the best solution seen so far);
2  $k = 0$ (with k the number of iterations);
3  while $k <$ maximum number of iterations and $f(s^*) \neq 0$ do
4      $k = k + 1$;
5      if $k \bmod 3 \neq 0$ then
           /* perform single moves of one exam to another time slot       */
6          Generate set of solutions $V^* \subseteq N(s, k)$ (set of neighbours of s after single move);
7          Sort $V^*$ by ascending $f(s')$ and select the best $s'$;
8          foreach solution $s'$ in $V^*$ do
9              if not $tabu(s')$ or $f(s') < f(s^*)$ (aspiration criterion) then
10                 $s = s'$;
11                 break;
12             end
13         end
14     else
           /* swap entire period                                           */
15         Generate set of solutions $V^* \subseteq N(s, k)$ (set of neighbours of s after swapping periods);
16         Sort $V^*$ by ascending $f(s')$ and select the best $s'$;
17         $s = s'$;
18     end
19     if $f(s') < f(s^*)$ then
20         $s^* = s'$;
21     end
22 end
23 return $s^*$
```

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 p_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

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

5.4 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. Firstly, the objective function originally defined in the initialisation phase as

$$f(s) = \sum_{i=1}^{N} \Big(\sum_{j,k \in E_i} p x_{jk} \Big) \tag{4}$$

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

$$f(s) = \sum_{i=1}^{N} \Big(\sum_{s \in S(E_i)} p^{c_{si} - 1} \Big) \tag{5}$$

with N 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_{si} the number of exams student s has on period i, and p the weight of conflicts. By placing c_{si} 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 resulted often resulted in the scenario that the same exam was continuously being moved. since this generally was one of the largest exams, it always has a large impact on the objective function and kept being determined 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.

5.5 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.

6 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. 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

Tabu search has a limited amount of hyperparameters compared to other optimisation methods. This aids in the parameter tuning process. 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 its 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

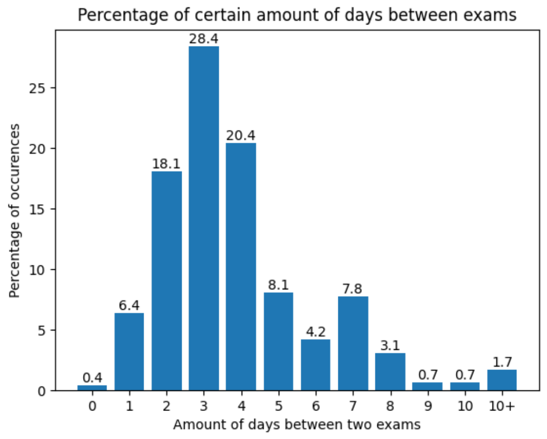
	Parameter	**Description**
All versions		
	P_INITIALISATION	weight of conflicts during initialisation phase
	P_OPTIMISATION	weight of distance between exams
	MAX_ITER_INITIALISATION	max amount of iterations during initialisation phase
	MAX_ITER_OPTIMISATION	max amount of iterations during optimisation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

7 Results / Evaluation

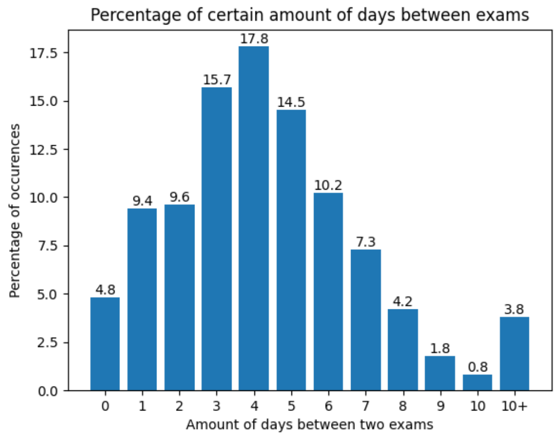
In order to evaluate the results from the experiment, we have to verify whether the phases perform their objective as described. Firstly, does the initialisation phase succeed in generating a ’feasible’ solution namely one without or with a minimal number of hard constraint violations? Secondly, does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

7.1 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 and showcase the priorities. 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 3+ days in between exams as much as possible. This distribution is especially visible in the schedules for the faculty FTI. In order to be considered a superior solutions, 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.

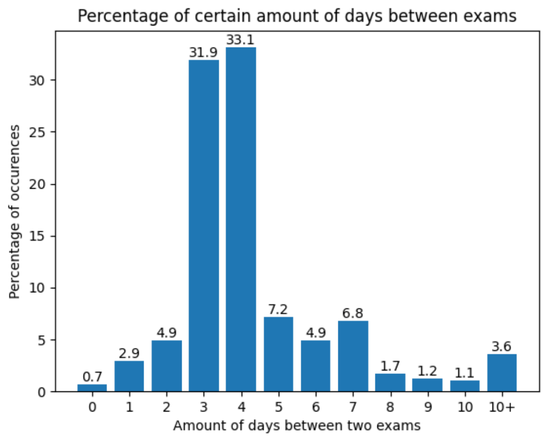


(a) FTI

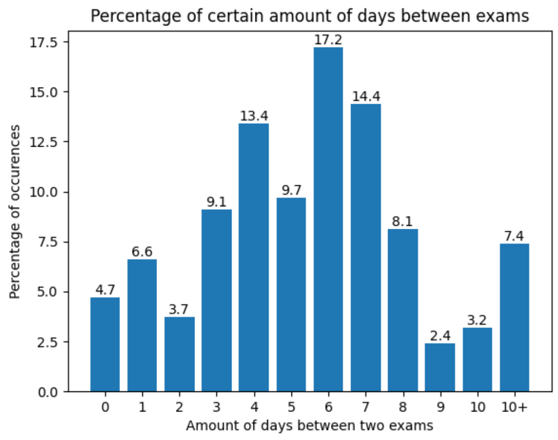


(b) FWET

Figure 4: Manual exam distribution for January 2021



(a) FTI



(b) FWET

Figure 5: Manual exam distribution for June 2021

7.2 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 1 day.

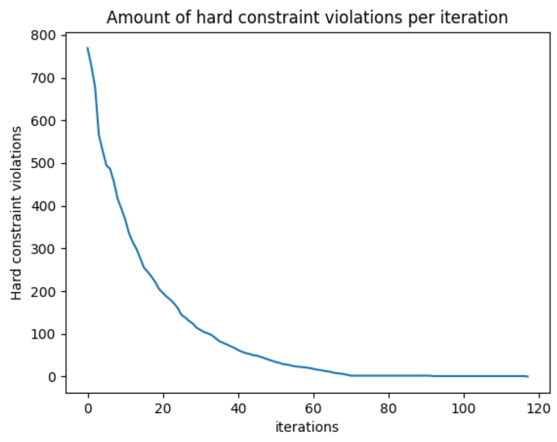
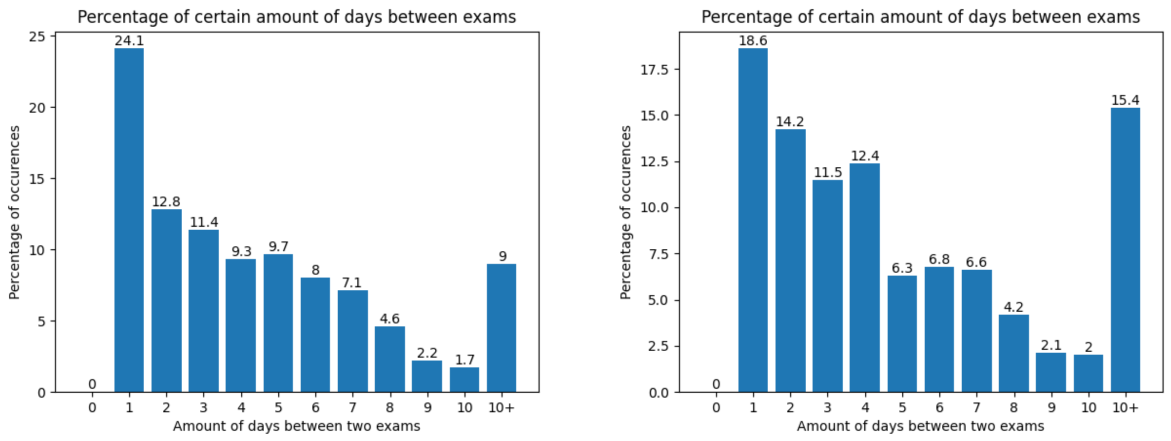


Figure 6: Hard constraint violations



(a) FTI

(b) FWET

Figure 7: Exam distribution after initialisation phase for June 2021

7.3 Optimisation phase

While we have shown that tabu search can efficiently generate a feasible solution from the provided data set, the distribution will determine whether the automated schedules are able to beat the manual schedules.

8 Threats to Validity

9 Conclusion

References

[1] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science (New York, N.Y.)*, 220:671–80, 06 1983. pages 10

[2] 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 10

[3] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992. pages 10

[4] 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 10

[5] Hussein Abbass. Mbo: marriage in honey bees optimization-a haplometrosis polygynous swarming approach. volume 1, pages 207 – 214 vol. 1, 02 2001. pages 10

[6] Simon Kristiansen, Matias Sørensen, and Thomas Stidsen. Integer programming for the generalized high school timetabling problem. *Journal of Scheduling*, 18, 08 2015. pages 10

[7] 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 10

[8] Esra Aycan and Tolga Ayav. Solving the course scheduling problem using simulated annealing. pages 462 – 466, 04 2009. pages 11

[9] 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 11

[10] 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. 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), PATAT ; Conference date: 28-08-2012 Through 31-08-2012. pages 11

[11] N. Pillay and W. Banzhaf. An informed genetic algorithm for the examination timetabling problem. *Applied Soft Computing*, 10(2):457–467, 2010. pages 12

[12] Shu-Chuan Chu and H.L. Fang. Genetic algorithms vs. tabu search in timetable scheduling. pages 492 – 495, 01 2000. pages 12

[13] Ruey-Maw Chen and Hsiao-Fang Shih. Solving university course timetabling problems using constriction particle swarm optimization with local search. *Algorithms*, 6:227–244, 04 2013. pages 13

[14] 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 13

[15] 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 14

[16] Fred Glover and Eric Taillard. A users guide to tabu search. *Ann. Oper. Res.*, 41:1–28, 03 1993. pages 16

[17] Fred Glover and Manuel Laguna. *Tabu search I*, volume 1. 01 1999. pages 16

[18] 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 16