Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Econometrics and Operational research

A discrete Grey Wolf algorithm to create high quality personalized learning schedules fast

Sjoerd Hoff (480254)

Supervisor:	Danny Zhu
Second assessor:	prof. dr. D. Huisman
Date final version:	2nd July 2023

Abstract

In personalized learning, learners decide which learning activities they want to participate in. Though the concept has been proven effective, there are some difficulties implementing it in the real world. One of the main challenges is creating schedules of high quality in a short amount of time. The Hourly Learning Activity Planning Problem is a way to model schedule creation. We use different experimental settings to test the effectiveness of two solution methods for the problem: the proven Adaptive Large Neighborhood Search and the novel Grey Wolf algorithm, which we adapted to the personalized learning setting. The Grey Wolf algorithm shows great promise. With similar running times, it outperforms Adaptive Large Neighbourhood Search in 429 out of 576 instances. The improvement is largest in the hardest to solve instances, making it even more useful in practice.

1 Introduction

In traditional education large groups of students study the same materials, at the same speed, at the same times. This 'one-size-fits-all' approach to learning does not work optimally for students who deviate from the average. Therefore, in many places around the world, schools are moving towards personalized learning as their education system. The foundation of personalized learning is the view that the learner drives their own learning. In practice this is often done by having the learner decide which subjects they want to study, at which pace and at which level.

One of the logistical challenges schools face when moving towards personalized learning is scheduling lessons. Many studies have been done regarding scheduling of lessons in a teacher-class environment. The problem of making a timetable for all classes and teachers is NP-complete, see Asratian and Werra (2002). Depending on the restrictions it can even by NP-hard. Many different methods to solve these problems have been developed, both exact and heuristics. Scheduling for personalized learning is an even more complex problem. This is due to the learners having choice in which modules they follow. There are more possibilities to take into account when trying to make a schedule. This difficulty holds schools back in implementing some form of personalized learning.

One of the specific problems seen in practice is the short-term changes that sometimes need to be made to schedules. This is necessary when, for example, a teacher calls in sick or, more specific for the personalized learning setting, a student changes their learning demands. In these cases, an updated schedule needs to be made quickly - often in the span of a few minutes up to an hour. Wouda, Aslan and Vis (2023) have introduced the Hourly Learning Activity Planning Problem (HLAPP) to model the creation of a one hour schedule according to learner demands. Solving to optimality is possible, but takes too much time. Therefore they introduced an Adaptive Large Neighborhood Search (ALNS) metaheuristic to solve the problem of creating a one hour schedule quickly.

In similar research on the scheduling of nurse shifts, many solving methods have been explored to solve that problem, see Ngoo, Goh, Sabar et al. (2022). It has been observed that different solution methods perform better than ALNS for the nurse-rostering problem. A logical extension of previous research would therefore be to test these better performing solution methods on the problem of finding schedules for personalized learning. We adapt the Grey Wolf

Optimization algorithm to our specific problem. Grey Wolf Optimization has mostly been applied to continuous optimization problems. In this paper we deal with a discrete problem. The algorithm needed some adaptations to work on discrete problems, for which we used the paper by Ngoo, Goh and Kendall (2022) as a starting point. They adapted the Grey Wolf algorithm to work for the nurse-rostering problem.

We make modifications to their algorithm, to make it work for the personalized learning scheduling problem. New mutation operators and a novel crossover operator are introduced. We test our algorithm on generated data, with many different experiment parameters. From medium school sizes to extra large, from small demand spreads to large and other differences. Our algorithm produces better solutions than the ALNS metaheuristic in 429 out of 576 instances. This is with a runtime set to 60 seconds, well within the 10 minutes set as reasonable timelimit for the re-scheduling problem. Furthermore, the algorithm performs well across the board, whereas ALNS mainly performs well for instances with small demand spreads. Since the Grey Wolf algorithm obtains the best results allround, and gives the best results in the most difficult to solve instances, these results show great promise for use in practice.

In the next Section a literature review of relevant papers on scheduling and the Grey Wolf algorithm is given. Section 3 describes the hourly learning activity planning problem. In Section 4 we explain the methodology used to solve the problem. The results that we obtained are presented in Section 5. Finally, Section 6 concludes and discusses the findings.

2 Literature

This literature review explores previous research in the field of scheduling, focusing on two main categories: personalized learning and traditional school timetabling. Within personalized learning there have been a few studies that address similar problems to the Hourly Learning Activity Planning Prolbem (HLAPP) as discussed by Wouda et al. (2023). In the realm of traditional scheduling, a lot more research has been conducted. A common theme is the use of heuristics to find computationally efficient solutions. Also, an overview of previous research into the Grey Wolf Algorithm is given. The algorithm has had great results in the realm of continuous optimization. The method has not been applied to many discrete problems, though a few cases can be mentioned.

First of all, a few studies have been done in the field of scheduling for personalized learning. Practical research has been done by Santiago et al. (2005). They presented a two-phase heuristic to obtain schedules personalized for learners in Spain. A similar practical research was conducted by Kristiansen, Sørensen and Stidsen (2011), where the planning of elective courses within traditional time-tables is researched. In their paper the problem is modeled using integer programming and exact solution methods are proposed, including a Branch-and-Price framework using partial Dantzig-Wolfe decomposition. In their research on scheduling for personalized learning, Kannan, van den Berg and Kuo (2012) study a very similar problem to ours, which is completely centered around the learner. They developed an algorithm that splits the problem into sub-problems, and then solves it by using heuristics. However, reported computation times are still very large. Aslan, Bakir and Vis (2020) researched scheduling for personalised learning with a demand driven approach. They use a hyper-heuristic to generate week-long schedules. The performance of the hyper-heuristic is compared to a commercial solver, showing promising results. Finally Wouda et al. (2023) did research on the same demand driven personalized learning situation, however, viewed as a one-hour scheduling problem. Their output is a schedule for one hour. The main practical use is suggested to be for short-term changes to a longer time-period schedule. In their research they used the Adaptive Large Neighbourhood Search (ALNS) meta-heuristic for solving. The meta-heuristic proved significantly more computationally efficient than solving exactly.

Compared to personalized learning, scheduling for 'traditional' schools has seen a bit more research. Pillay (2013) gives an overview of studies done in this field up to 2013. We can see that many different solution methods have been developed for this problem, mostly heuristics. Due to the different datasets used, it can be difficult to compare solution methods directly. A clear conclusion that can be made from the research is that heuristics are the only computationally efficient way to solve school timetabling problems. The research does not agree on which heuristic is best. Two later studies are of particular interest. Kiefer, Hartl and Schnell (2017) applied an ALNS alogirthm, with great succes. Veenstra and Vis (2016) researched a re-rostering problem, in a traditional rostering setting. In their case computation time for the optimization method was crucial. They found that a heuristic was most efficient in this regard.

Many of these heuristics exist in the literature. Grey Wolf optimization is one of these, with promising results. First introduced by Mirjalili, Mirjalili and Lewis (2014), it has since been extended by researchers and applied to many different problems (Almufti, Ahmad, Marqas & Asaad, 2021). The algorithm often produces near-optimal results, with short computation times. Al-Betar, Awadallah, Faris, Aljarah and Hammouri (2018) have improved the algorithm by changing the selection methods to be less greedy. Nadimi-Shahraki, Taghian and Mirjalili (2021) made improvements in the neighbourhood searching of the algorithm, giving it more exploratory freedom. Ngoo, Goh, Sabar et al. (2022) has adapated Grey Wolf for the nurse-rostering problem, which has many similarities with the HLAPP model. In their research they find that it performs better than six other methods that have been applied to their dataset. Due to the proven performance in many applications, we adapt the Grey Wolf algorithm to the HLAPP problem.

3 Problem and model

In a generalized personalised learning setting, there are four resources that need to be allocated. There is the set of learners $l \in L$ who demand certain learning activities. Similarly we have the set of teachers $t \in T$, which can oversee learning activities. Next to that, the topics which can be taught in learning activities are included, called modules $m \in M$. The last resource is the classroom $c \in C$, which are the places where learning activities are scheduled. All these resources are assumed to be finite.

The learners l have demands $D_{lm} \in R_{\geq 0}$ of all modules m. There are two types of modules, the 'instruction' modules, where a certain module is actively taught, and a 'self-study' module, where a learner works on their most-demanded module under the supervision of a teacher, but no active teaching. Schools generally prefer active teaching, therefore self-study of a module does not meet the demand of the learner fully. The difference is modeled by a self-study parameter w

$(0 \leq w \leq 1)$, which gives a penalty to the demand met by the self study activity. The demand for self-study of a learner is given by D_{lm_S}, where m_S is the module for self-study. The self-study demand is given by $w * max_{m \in M \setminus m_S} D_{lm}$. The w parameter sets the penalty for self-study. The demand is zero for modules that the learner is not eligible for.

Teachers t are not capable of teaching all modules m. This relationship is modeled through a binary qualification matrix $Q^T_{tm} \in \{0, 1\}$. When teachers are qualified to teach module m, Q^T_{tm} takes on the value 1, and 0 otherwise. All teachers are qualified to teach the self-study module.

Not all classrooms c can be paired with all modules m. For example, a gym class cannot be held in a normal classroom and vice-versa. This relationship is also modeled through a binary qualification matrix $Q^C_{cm} \in \{0, 1\}$. Classrooms also have a hard capacity in the form of seating space, given by N_c.

The learners are paired with modules through the decision variable y_{lm}, which takes on the value 1, if the learner is scheduled for module m and 0 otherwise. Furthermore, activities are modeled by the decision variable x_{mct}, which takes on the value of 1 when module m will be taught in classroom c by teacher t and 0 otherwise. There are two additional parameters which can be set, to form additional hard constraints for the scheduled activities. The parameter δ^- sets a minimum to the number of learners in an activity and δ^+ sets a maximum to the number of learners in an activity.

The complete model is given by:

$$\max \quad \sum_{l \in L} \sum_{m \in M} D_{lm} y_{lm} \tag{1}$$

$$\text{subject to:} \quad \sum_{l \in L} y_{lm} \leq \sum_{c \in C} \min(\delta^+, N_c) \sum_{t \in T} x_{mct} \qquad \forall m \in M \setminus \{m_S\} \tag{2}$$

$$\sum_{l \in L} y_{lm_S} \leq \sum_{c \in C} N_c \sum_{t \in T} x_{m_S ct} \tag{3}$$

$$\sum_{l \in L} y_{lm} \geq \delta^- \sum_{c \in C} \sum_{t \in T} x_{mct} \qquad \forall m \in M \tag{4}$$

$$\sum_{m \in M} y_{lm} = 1 \qquad \forall l \in L \tag{5}$$

$$\sum_{m \in M} \sum_{c \in C} x_{mct} \leq 1 \qquad \forall t \in T \tag{6}$$

$$\sum_{m \in M} \sum_{t \in T} x_{mct} \leq 1 \qquad \forall c \in C \tag{7}$$

$$\sum_{c \in C} x_{mct} \leq Q^T_{tm} \qquad \forall t \in T, \forall m \in M \tag{8}$$

$$\sum_{t \in T} x_{mct} \leq Q^C_{cm} \qquad \forall c \in C, \forall m \in M \tag{9}$$

$$y_{lm} \leq \mathbb{1}_{D_{lm} > 0} \qquad \forall l \in L, \forall m \in M \tag{10}$$

$$y_{lm} \in \{0, 1\} \qquad \forall l \in L, \forall m \in M \tag{11}$$

$$x_{mct} \in \{0, 1\} \qquad \forall m \in M, \forall c \in C, \forall t \in T \tag{12}$$

The objective (1) is the sum of all the demands met by the assignment of learners. The goal is to have as much demand met as possible. The first set of constraints (2) ensures that the number of learners in an activity does not exceed the classroom limits, or the set maximum amount by the school. Constraint (3) does the same for self-study activities, where the maximum limit set by the school does not apply. The next set of constraints (4) applies the minimum activity size, such that no activities are made with less learners than the minimum size. With constraints (5) it is ensured that every learner is assigned to a module. Constraints (6) ensures that classrooms are only assigned to at most one activity. Constraints (7) do the same for teachers. Constraints (8) and (9) make sure that each teacher and classroom are suitable for assigned activities. Constraints (10) ensures that learners are only assigned to modules they are eligible for, so for which $D_{lm} \geq 0$. Lastly, constraints (11) and (12) set the y and x variables to binary variables.

Solving this problem does not lead to a schedule directly, for that an extra step is necessary. Learners are assigned to modules by y and there are enough teachers and classrooms to host those modules with learners ensured by x. To get a schedule, first all x activities are filled to the minimum δ^-, then the remaining learners are added to activities with their respective module. When there are multiple activities of a module, one is filled until it is full (by δ^+), then the second one gets filled, etc. The order in which the activities is filled is by activity number. In this way a schedule of activities is obtained.

4 Solution methodology

In this section we explain the methods used to obtain solutions to the HLAPP problem. We first discuss the Adaptive Large Neighbourhoud Search method, introduced by Wouda et al. (2023). The exact same methodology is used as in their paper. Next we introduce the Discrete Grey Wolf algorithm, adjusted for the HLAPP problem. A Discrete Grey Wolf algorithm used in nurse-rostering is used as a starting point, with small changes made. Custom mutation operators and a custom crossover operator are introduced as well.

4.1 Adaptive Large Neighbourhoud Search

A general framework is given by Pisinger and Ropke (2010). This framework is used to create the Adaptive Large Neighbourhood Search used here. The ALNS described is that of Wouda et al. (2023). We cover the main parts of the method, being the algorithm itself, the intial solution, destroy operators, repair operators, acceptance criterion, local search and operator selection. For a full description we refer to Wouda et al. (2023).

The ALNS algorithm start with an initial feasible solution s.This solution should be a feasible solution, however it does not need to be a solution of high quality. Then the algorithm iterates for a fixed number of iterations. In each iteration, the current solution is destroyed by a destroy operator and repaired by a repair operator. The operators are chosen by a roulette wheel. These operations create a new solution, which is evaluated. The roulette wheel is updated based on this evaluation. If it is the new best, a local search procedure is run, the output of which is set as the new best solution. In the end the best solution is returned.

Algorithm 1 Adaptive Large Neighbourhood Search

```
Input: Initial feasible solution $s$
Output: Best observed solution $s^*$
  $s^* := s, \rho_D := (1, ..., 1), \rho_R := (1, ..., 1)$
  repeat
      Select destroy and repair methods $d_{op} \in O_D, r_{op} \in O_R$ using $\rho_D$ and $\rho_R$.
      $s^c := r_{op}(d_{op}(s))$
      if $s^c$ is accepted then
          $s := s^c$
      end if
      if $s^c$ has a better objective value than $s^*$ then
          $s^* :=$ Local-Search($s^*$)
          $s := s^*$
      end if
      Update $\rho_D$ and $\rho_R$
  until maximum number of iterations is exceeded
  return $s^*$
```

4.1.1 Initial solution

An initial solution is created by assigning all learners to self-study. The activities are scheduled by selecting a random teacher (all teachers are qualified for self-study) t and suitable classroom c. The classrooms are then filled up to their limit N_c. In our experiments it is impossible for this initial solution to be infeasible, due to the amount of learners always being smaller than the amount of learners the school can accommodate, and also enough teachers being available to supervise. In practice it also very unlikely, since it requires a school to have more students than it has space and teachers for them.

4.1.2 Destroy operators

The destroy operators remove part of the solution. Their goal is to remove around d learners from their assignments.

Random activity removal:
This operator removes an entire activity from the solution. All learners assigned to that activity, the corresponding teacher and classroom are set to unassigned. This procedure repeats until at least d learners are removed.

Smallest activity removal:
The smallest activity is removed from the solution by this operator. Just like with random activity removal, the learners, teacher and classroom are set to unassigned. The process is repeated until at least d learners are removed.

Random learner removal:
This operator removes a random learner from an activity. The learner is set to unassigned. It only does this move if the solution remains feasible, so no learners are removed that make the

activity size too small (smaller than δ^-). This process is repeated until d learners are removed, or no more learners can be removed feasibly.

Worst regret learner removal:
This operator removes a set of learners that are the least met in their demands, seen as regret. Regret is the difference between the demand met by their current assignment, and the maximum demand that could be satisfied by a different assignment. All learners are sorted by their regret. A total of d learners are selected to be removed. If the removal of a learner would result in an infeasible solution, this learner is not removed. The d learners are selected using a skewed distribution, which favours larger over smaller regrets.

4.1.3 Repair operators

The repair operators put the unassigned learners from the destroy operators in new assignments. Solution feasibility is retained.

Break-out activity:
The unassigned learners are grouped by the modules they demand. The modules are sorted by decreasing total learner demand. A module is scheduled, if δ^- is respected and a suitable teacher and classroom is available. The learners are then assigned to this module. If the module is a second-degree module, a second-degree teacher is preferred over a first-degree teacher. Second-degree meaning modules that are taught in the first three years of school, and a second-degree teacher is only able to teach these modules, not modules from the last three years. A more detailed explanation of second-degree and first-degree can be found in Section 5.1. The smallest classroom that fits the learners is also preferred. If however the amount of learners is greater than the maximum available classroom size, the largest classroom is chosen, to fit as many learners as possible into the activity. In that case not all the learners are assigned to this module. Next to that, this operator attempts to assign self-study learners into the new activities, when that leads to a better solution.

This process is continued until all possible activities are scheduled. If there are learners left, they get assigned according to the greedy learner insert, as seen below.

Greedy learner insert:
This operator takes a random unassigned learner, and assigns it to the best available learning activity. The solution has to remain feasible, so activities need to be smaller than $min\{\delta^+, N_c\}$ and the learner needs to demand the activity. If there is no available activity for the learner, they are assigned to self-study. If there is no self-study activity available, or it is full, a new self-study activity is created. In the rare case that there are no teachers or rooms available for self-study, the learning activity with the smallest demand met is removed, to make room for a new self-study activity. The learners from the learning activity also get assigned to the self-study activity.

4.1.4 Acceptance criterion:

A new solution is not always accepted. The process to determine acceptance is Simulated Annealing (SA) as described in Wouda et al. (2023). Solutions that improve the current solution are always accepted. If they are not better, they have a chance to get accepted, in order to increase the exploration power of the algorithm. The worse the solution, the lower the chance. Also, the more iterations have passed, the smaller the chance.

4.1.5 Local Search:

When a new best solution is found, Local Search is applied to improve the solution further. The local search operator first determines all the possible assignment moves that can be made for all learners, given the scheduled activities. It then applies the move with the best objective gain, if feasible. This process is repeated until no more moves can be applied that improve the solution.

4.1.6 Operator selection:

We have multiple Repair and Destroy operators. A Roulette Wheel is used to determine which repair and destroy operator should be used for an iteration. Each operator is given a weight, which determines the chance that it is chosen. The weights are updated at each iteration, according to their performance. The better performing operators get selected more often. The method is explained further by Wouda et al. (2023).

4.2 Grey Wolf Optimization

the Grey Wolf algorithm is inspired by nature. Grey wolfs are considered apex predators, meaning that they are at the top of the food chain. They hunt together in packs. An important rule of the pack is the social hierarchy. There is an alpha (leader), beta (second in command), and the rest are delta wolves. There is also a omega wolf, which could be considered the scapegoat of the pack. In general, a Grey Wolf algorithm attempts to mathematically model the social hierarchy and hunting techniques of grey wolfs, to 'hunt' for a best solution. For more details on this behaviour, and the original Grey Wolf algorithm for use in continuous problems, we refer to Mirjalili et al. (2014).

We use the improved Discrete Grey Wolf Optimization algorithm by Ngoo, Goh, Sabar et al. (2022) as a starting point. We have adapted this algorithm to the HLAPP problem. It can be seen below in Algorithm 1. It general it works as follows, with more details being provided in later sections. First, an initial solution is generated, see Section 4.2.1. A population of grey wolfs is initialized, with each receiving the initial solution as position. Then, the positions of each wolf get updated by applying mutation operators (see Section 4.2.2) until a number γ of learners is mutated. A mutation operator is chosen with equal chance. Each mutation operator mutates a random number ζ of learners. To obtain ζ, we draw from $U(0.25, 0.75)$. This value gets multiplied by γ to find the fraction of γ to mutate with this calling of the mutation operator. After finishing mutating, the objective value for each wolf is calculated, and the X_α, X_β and X_δ get updated accordingly. If the X_α solution is better than the best solution, the best solution gets updated to the X_α solution. A new position for all wolfs is generated by using the three best solutions (X_α, X_β and X_δ). A crossover operator is applied which combines these solution to create a new solution. If the new solution is better than the best solution, the best solution gets updated. This new solution is given as a position to all wolves, and the process is repeated until the time limit is reached. When the time limit is reached, the best solution at that point is returned.

Algorithm 2 Discrete Grey Wolf Optimization

```
Input: Generate an initial solution $s$
  Initialize a population of grey wolves, P = $X_1$, $X_2$, $X_3$ ... $X_n$, with solution $s$
  $X_\alpha = X_1$
  $X_\beta = X_2$
  $X_\delta = X_3$
  $s^* = X_1$
  repeat
      for Each $X_w$ in P do
          repeat
              Randomly select mutation operator with equal chance
              Apply mutation operator, with $\zeta$ as number of mutations.
          until mutations $\geq \gamma$
      end for
      Sort the Wolfs by objective value, update $X_\alpha$, $X_\beta$ and $X_\delta$.
      if $X_\alpha > s^*$ then
          $s^* = X_\alpha$
      end if
      Generate a new position $s_n$ using Discrete Blend Crossover($X_\alpha$, $X_\beta$, $X_\delta$)
      if $s_n > s^*$ then
          $s^* = s_n$
      end if
      Assign $s_n$ to all $X_w$ wolves in P.
  until time > time limit
  return $s^*$
```

4.2.1 Initial solution

The Grey Wolf crossover operator does not respond well to starting with the feasible solution of ALNS, it performs better with an initial solution of higher quality. Therefore we generate an initial solution as follows: All learners are set as unassigned. Then the repair operator of Section 4.1.3: 'break-out activity' is applied. This results in a solution with many learners already in instruction activities.

4.2.2 Mutation operators

To create random changes in the solutions, mutation operators are used. They are inspired by evolution, where random genetic modifications lead to long term changes of species. Similarly, here random mutations are applied to solutions. If they improve the solution, there is a high probability they are carried over to later solutions. Two mutation operators are proposed:

Mutation 1: Learner reassignment
This operator removes ζ random learners from their assignment. These learners then get reassigned to other, or new activities, in the same way as the break-out operator of section 4.1.3.

Mutation 2: Activity reassignment

This operator removes a random activity, setting all learners to unassigned. This process is repeated until at least ζ learners are unassigned. Then these learners get reassigned with the break-out operator.

4.2.3 Crossover operator

We introduce a crossover operator loosely based on the Blend Crossover introduced by Eshelman and Schaffer (1993). The pseudecode can be seen in Algorithm 3. The crossover operator start with three solution to combine, the X_α, X_β and X_δ wolfs. The alpha wolf is set to be the new solution. For each learner l, it is checked what module m they are assigned to. If the assignment is agreed upon by all three input solutions, the assignment is kept, otherwise the learner is set to unassigned. After doing this for all learners l, the assigned activities are checked if they meet the minimum activity size requirement (δ^-), since it is possible assignments have gotten to small. First it is attempted to add learners from the unassigned set until the minimum activity size is met. Only learners whose demand for the module (D_{lm}) is bigger than their demand for self-study (D_{lm_S}) are assigned. If there are not enough learners in the unassigned set to meet the minimum activity size, the activity is removed and the learners are set to unassigned. Finally, all unassigned learners are added back in to the new solution by greedy learner insert, as seen in Section 4.1.3. Finally the newly made solution is returned.

Algorithm 3 Discrete Blend Crossover

```
Input: Three parent solutions: $X_\alpha$, $X_\beta$ and $X_\delta$
Output: New solution $X_w$
  for Each learner $l$ do
      if $y_{lm}$ is the same for $X_\alpha$, $X_\beta$ and $X_\delta$ then
          Set $y_{lm}$ of $X_w$ same as $X_\alpha$, $X_\beta$ and $X_\delta$.
      end if
      if $y_{lm}$ is not the same for $X_\alpha$, $X_\beta$ and $X_\delta$ then
          Add learner to the set unassigned $U$
      end if
  end for
  for activity in $X_w$ do
      if # learners in activity $< \delta^-$ then
          repeat
              for learner $l \in U$ do
                  if activity is preferred over self-study then
                      Add learner to activity
                  end if
              end for
          until learners in activity $= \delta^-$ OR all learners checked
      end if
      if # learners in activity $< \delta^-$ then
          remove activity
          Add learners of activity to $U$
      end if
  end for
  Place all unassigned learners in $X_w$ using greedy learner insert
  return $X_w$
```

5 Results

In this section we present our results. We first start with the experimental design. It is explained what experiments are run and which parameters differ between experiments. Next, it is explained how the results have been obtained. In Section 5.2.1 the performance of the solution methods are compared against the exact solver and each other. Optimality gaps, the number of learners in instruction activities and runtimes are discussed. Finally in Section 5.2.2 the effects of changing experiment parameters on solution quality is discussed. We compare these results with results found by Wouda et al. (2023).

5.1 Experimental design

The data is supplied by Wouda et al. (2023). We run the same 144 experiments, all a different permutation of the parameters seen in Table 1. This gives a broad spectrum of different school parameters, that influence the size and difficulty of the problem. There are also parameters that schools themselves can influence. The self-study parameter and instruction classrooms and capacities can be changed very easily by schools. Schools also have some influence over the teacher qualification distribution.

Table (1) The experiment parameters

Parameter	Levels
Self-study penalty parameter (w)	50%, 75%
Demand spread (sigma)	0, 1, 2, 3
School size (# learners)	800, 1200, 1600
Teacher qualification distribution	(1;0;0), (0.5;0.5,0), (0.4;0.4;0.2)
Instruction classrooms and capacities	Regular number of classrooms of 32 capacity, double the number of classrooms of 16 capacity
Minimum activity size	5
Maximum instruction activity size	30

For each experiment, we use the first 4 instances created by Wouda et al. (2023). Each instance has defined learners, modules, teachers and classrooms, with corresponding demands, qualifications and activity size constraints. All instances have activity size constraints of 5 as a minimum and 30 as a maximum (δ^- and δ^+ respectively). Each regular classroom has a capacity of 32 learners, and the large self-study classrooms a capacity of 80 learners. There are 576 modules in each instance, made up of 12 courses, with each 48 modules, nominally 8 per year of study.

Three different school sizes are considered:

- **Medium (M):** 800 learners, 80 teachers, 40 instruction classrooms, 3 large self-study classrooms
- **Large (L):** 1200 learners, 120 teachers, 60 instruction classrooms, 4 large self-study classrooms
- **Extra Large (XL):** 1600 learners, 160 teachers, 80 instruction classrooms, 6 large self-study classrooms

It is likely that group sizes in personalized learning are smaller than in traditional education. Due to the ability for learners to follow different modules from the same course at the same time, groups will become smaller. Therefore, schools could choose to place walls in all their classrooms, splitting them in half. It would double the number of classrooms, while cutting the maximum capacity in half to 16. Half of the experiments have this policy applied.

The 12 different courses do not get equal learning hours in practice. The number of teachers per course should also not be equal. We set teacher qualifications to the same fraction as weekly hours spend by learners on a specific course. To give an example, if the course English is on average taught three hours a week, with a 30 hour schedule, 10% of teachers will be English teachers.

Apart from that, teachers are also split into three different qualifications: first-degree, second degree and third degree. A first degree teacher can teach all modules from the course they teach. A second degree teacher can only teach students from the first three years of education. A third degree is not qualified to teach, however, they can supervise a self-study activity. Three distributions are examined here: (1;0;0), where all teachers are first-degree, (0.5;0.5;0), where half of the teachers are first-degree and half are second-degree and (0.4;0.4;0.2), where 40% of teachers are first-degree, 40% are second degree and 20% of teachers are third-degree. In economic sense, first-degree teachers are almost always more expensive than second-degree teachers, and second-degree teachers more expensive than third-degree teachers. We test if different distributions of teacher qualifications has a significant impact on solution quality.

The learner demands are generated as follows. A random demand is generated for each course. First, a module to demand is selected. The point where the learner should be following nominal progression is obtained by taking the midpoint of the year they are in. This is computed by taking the year they are in $y_l \in \{0, 1, ..., 5\}$ and applying the following function: $\mu_l = 8y_l + 4$. Then, to introduce randomness in the selected module, we draw from $N(\mu_l, \sigma)$, where σ differs between experiments, and takes on one of the values from the demand spread of Table 1. A higher value of σ will thus give a bigger demand spread. The result gets rounded to the nearest integer in $\{1, 2, ..., 48\}$, which is then selected as the module this learner will demand. The actual demand value is drawn from an $Exp(\beta = 2)$ distribution, which ensures non-negative values.

5.2 Results

The model presented in section 3 is programmed in Python 3.10. We use Gurobi to obtain exact solutions. The ALNS metaheuristic is also programmed in Python 3.10, and uses the open-source package ALNS, made by Wouda and Lan (2023). The Grey Wolf Algorithm too was programmed in Python 3.10 and uses parts of the ALNS metaheuristic.

For exact results, to compare the metaheuristic and Grey Wolf with, the exact results found by Wouda et al. (2023) were used. These results were publicly available. Reproducing these results ourselves took too much time and computation power. To solve, they used two Intel Xeon E5 2680v3 2.5 GHz CPU cores with 32 GB of memory. The solver was given 8 hours of runtime. This solved 88% of instances. The remaining instances were given another run on the two processors with 64GB of memory and two days time. After this 99% of instances were solved.

Both the ALNS and the Grey Wolf algorithm were solved using the M1 Pro processor with 16gb of memory for all instances. The metaheuristic was run with the parameters set to the values used by Wouda et al. (2023). They did extensive tuning to find optimal paremeters. They used a Bayesian optimisation package to do hyperparameter optimisation called SMAC3 (Lindauer et al., 2022).

We did some empirical testing for the parameters of the Grey Wolf algorithm. We found that setting the number of wolfs to 30 and the mutation rate to 40% produced good results. We set the parameter γ to always be half of the number of learners. So at each iteration, a large number of mutations are made. Also, the runtime was set to 30 seconds and 60 seconds. We noticed that longer runtimes than 60 seconds did not result in significantly better results in most instances. We did not perform hyperparameter optimization to optimize these parameters.

5.2.1 Performance of ALNS and Grey Wolf

We have taken the exact results of optimizing the ILP formulation of the problem as a baseline for comparing the performance of different solution methods. First of all, we have run an experiment where we gave the ILP 10 minutes of solving time, and took the best solution at this point. For this, we used the ILP results obtained by (Wouda et al., 2023). Important to note is that these results were obtained on a very powerful computer, and in most cases could not be obtained with our computer. We will compare these results with the ALNS and Grey Wolf methods. However, due to the different computers used, this might not a fair comparison. Table 2 gives an impression of the results that the M1 Pro with 16gb of memory would obtain in terms of optimality gaps after ten minutes for medium school sizes at demand spreads of 1, 2 and 3. As we can see in Table 2, the gaps are very small when run on our computer. This is in line with the results found by the researchers, since they also find very small gaps at small school sizes, increasing in demand spread. This leads us to think that the ILP gaps the researchers have found, can be directly compared with the results found of ALNS and Grey Wolf when run on our computer.

Table (2) The average of the optimality gaps of the ILP after ten minutes for a medium school size run on the M1 Pro processor with 16gb of memory.

Demand spread	Gap (%)
1	0.00052
2	0.00200
3	0.00849

In Figure 1 the performance of the different solution methods is shown, with the optimality gap as metric. The gap between the objective value of the solution and the optimal solution is measured. We can see that the ILP after ten minutes performs by far the best. In most cases it attains, or nearly attains the optimal solution value. The figure does not show the outliers however. There are a few cases where the gap is more than 12%. Another thing to note is that most computers won't be able to run the ILP, because a very large amount of memory is required for most instances. That is why the heuristics are necessary. We can see that the ALNS heuristic performs reasonably well, with the optimality gap averaging 2.2%. The Grey Wolf algorithm on average outperforms ALNS. We can see that it has a lower average gap, of 1.6% with 30 seconds of runtime and 1.3% with 60 seconds of runtime.. When running for 30 seconds, the Grey Wolf algorithm is best 413 out of 576 instances. When run for 60 seconds, it outperforms ALNS in 429 instances out of 576.

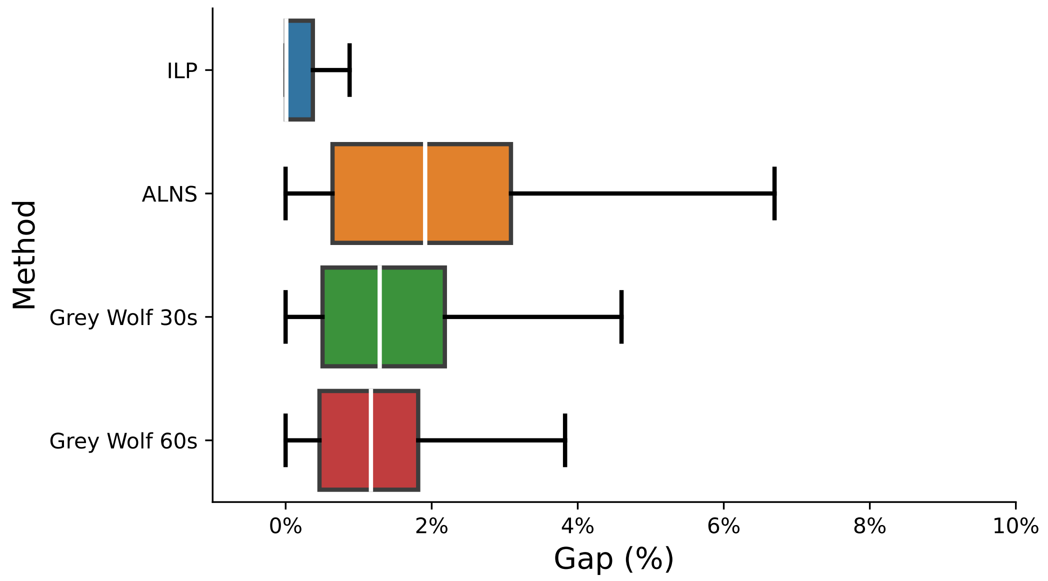

Figure (1) Optimality gaps of ILP after 10 minutes, ALNS and Grey Wolf after 30 seconds and 60 seconds.

Diving deeper, we can see that the best performing solution method differs by the parameters set for the experiment. In Figure 2 an overview is given of the optimality gaps for different school sizes and demand spreads. It is displayed as box-plots, in which the white line represents the mean, the boxes 25% of observed values on both sides of the mean and the whiskers the minimum and maximum gaps (excluding outliers).

For medium schools we observe that the ILP after 10 minutes in most cases arrives at or very near the optimal value. This is due to the smaller size of the problem. The ILP is fast enough in this case to solve to optimality in a relatively short time. We also see that the ALNS methods performs very well at a demand spread of 0. However, at higher demand spreads it is outperformed by Grey Wolf, by a large margin. Both on average and in terms of spread, the Grey Wolf has results closer to the optimal value at both 30 and 60 seconds of runtime.

At large school sizes we once again see that the ILP performs best, on average, in all cases. However, at larger demand spreads we do observe that at least 50% of instances have a gap larger than 0. ALNS once again outperforms Grey Wolf in the case of a demand spread of zero. At higher demand spreads Grey Wolf outperforms ALNS, the gaps are both significantly smaller and the maxima are lower.

The pattern continues for extra large schools. The ILP finds the optimal solution at demand spread zero. ALNS outperforms Grey Wolf at demand spread zero. At higher demand spreads we see that Grey Wolf outperforms ALNS, although, at demand spread one, the maxima are higher than that of ALNS. One thing we observe for the first time, is the Grey Wolf algorithm outperforming the ILP after 10 minutes. At demand spreads of two and three, we see that Grey Wolf has a smaller average gap and the maxima are also smaller.

In general we can see that at a demand spread of zero, the ILP always attains the optimal solution within 10 minutes. ALNS also performs well on these problems. The Grey Wolf method does not performs as well as ALNS, although the solutions remain close to the optimal value, so could still be viewed as acceptable. From demand spreads of 1 and above, we observe that Grey

Wolf outperforms ALNS. The average gap is smaller in all cases. One thing to note however, is that the maximum gaps are larger in a few cases, notably at large school size with a demand spread of 2 and 3 and extra large schools with a demand spread of 1. ILP after ten minutes outperforms both ALNS and Grey Wolf in most cases. However, when the problem gets to a certain complexity, Grey Wolf yields better results. We observe this at the extra large school size with a demand spread of 2 and 3.

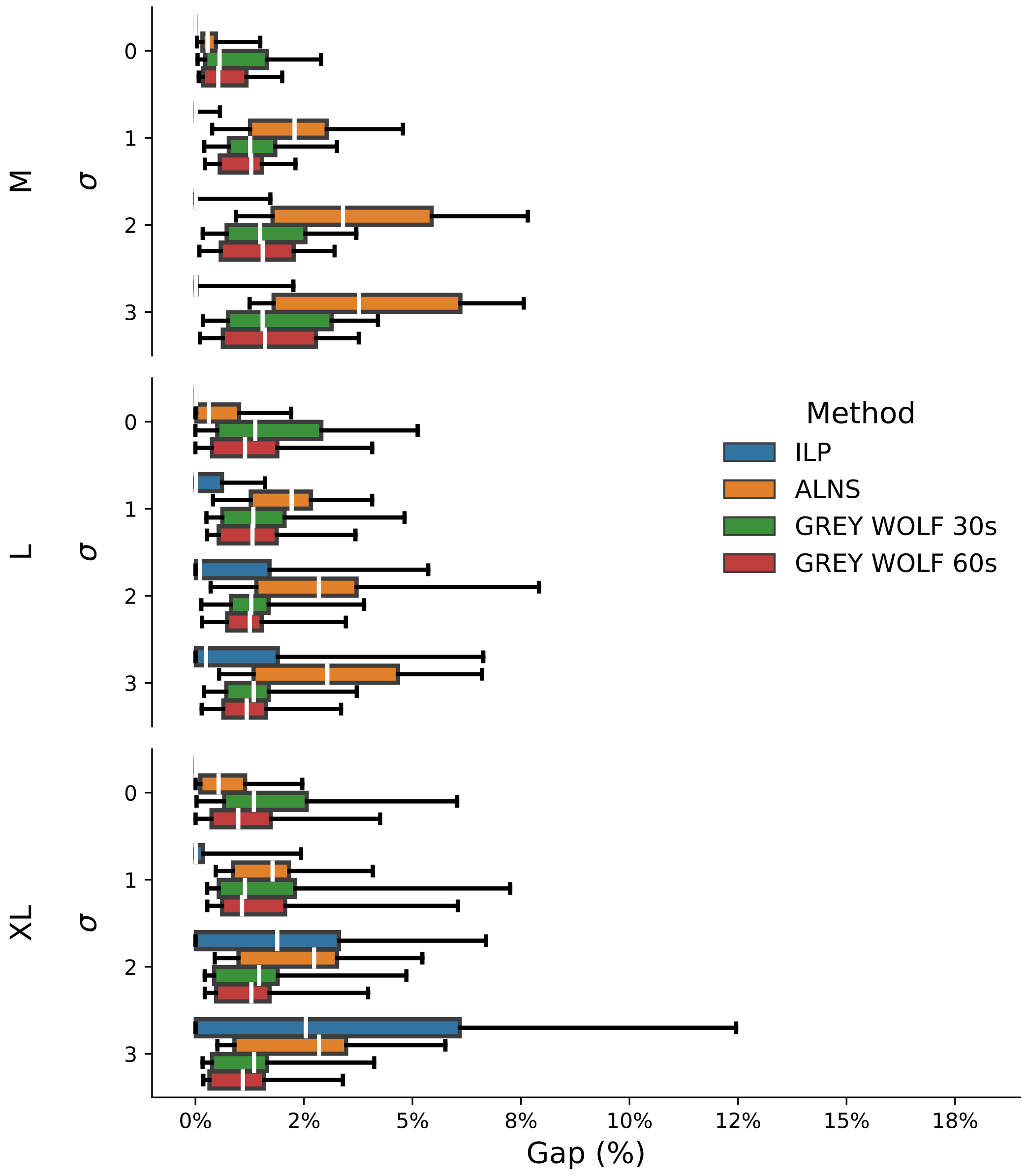

Figure (2) Optimality gaps of ILP after 10 minutes, ALNS and Grey Wolf after 30 seconds and 60 seconds, for each school size and demand spread.

Another, and in practice highly relevant, metric for solution quality is the number of learners in instruction activities. A school usually prefers as many learners in instruction activities as possible. In Figure 3 the number of learners in instruction activities in the optimal ILP solution is plotted against the same number for ALNS and Grey Wolf. A number as close to the ILP number is preferred. The results show that higher demand spread makes it harder to get close to the optimal solution. We observe a big difference between the optimal solution and ALNS in some cases. For example at the medium school size with demand spread 3, the ALNS has, on average, almost a hundred less students in instruction activities, a 30% decrease. Grey Wolf performs a lot better than ALNS in all instances, except at demand spread zero, where they both perform well. This is in line with the results seen in Figure 2. We see that the difference in optimality gap results in significant real-world differences: more students get assigned to instruction activities when using the Grey Wolf algorithm, which is desirable.

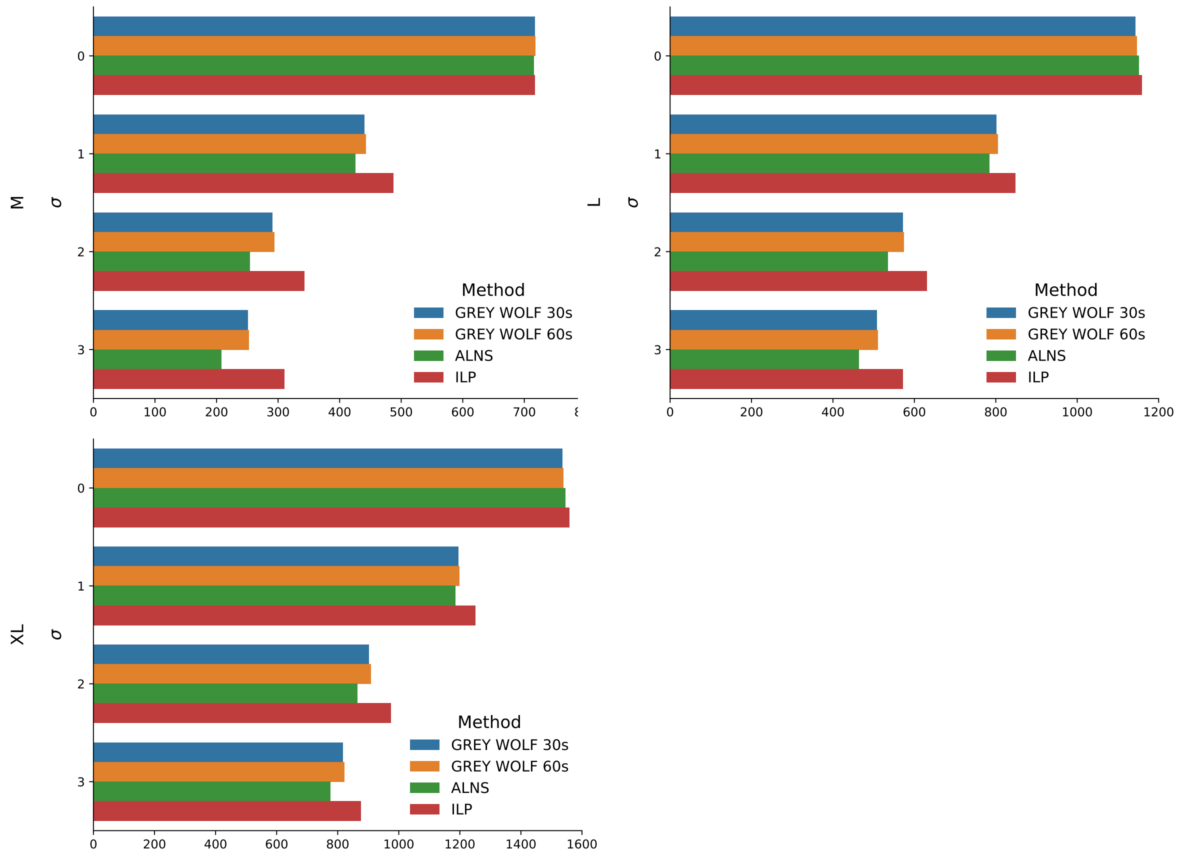

Figure (3) The solution quality compared to the optimal solution, in terms of number of learners in instruction activities.

Besides solution quality, runtimes are an important metric for performance. We know that the practical application of the HLAPP problem requires fast runtimes, preferably under 10 minutes. On our computer, with an M1 Pro processor and 16gb of memory, the runtime of the ALNS metaheuristic consistently stayed well below 10 minutes, even under 90 seconds. In Figure 4 the histogram of runtimes is shown. We can see that many instances are solved in around 30 seconds. We also see a peak at runtimes of a little less than a minute. This is also one of the reasons we ran the Grey Wolf algorithm with a runtime of both 30 seconds and 60 seconds, to give a fair comparison with ALNS. Also longer runtimes were empirically shown to lead only to small, if any, improvements to the solution.

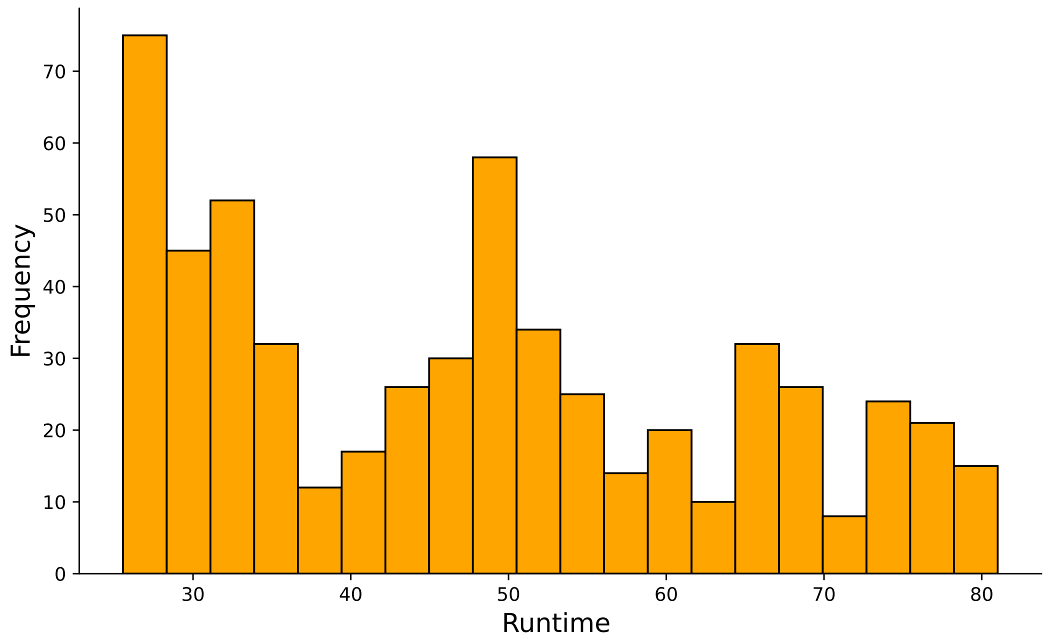

Figure (4) The runtimes of the ALNS metaheuristic.

5.2.2 Effects of school parameters

In Section 5.1 we outlined a few parameters schools can change and policy choices they can implement. In this section we investigate the effects of changing these parameter and choices on the best possible solution for the model. We use the results obtained by Grey Wolf in 60 seconds for this purpose and use the amount of learners assigned to self-study as a measure of schedule quality. A high number of learners assigned to self-study means that the model had difficulty scheduling large groups of learners for activities. We prefer to have as many learners as possible to be scheduled to an instruction activity. We will compare the results found with results obtained by Wouda et al. (2023). We expect the results to be similar, since the used data is the same. The only difference is the solution method. Wouda used the exact results (ILP) to obtain these figures, we use the results from the Grey Wolf algorithm. The figures can be found in Appendix A.

In Figure 9 shows that higher demand spread leads to a higher number of learners in self-study. This effect is especially pronounced for smaller schools. It is an expected effect of higher demand spread, since it leads to less people demanding the same module. Therefore it is harder to create large activities. The stronger effect for smaller schools is due to too few learners demanding the same module, making the possible activity size too small to schedule in some cases.

Next we will discuss varying the self-study penalty ω. Lowering the ω leads to a bigger penalty for self-study in comparison to instruction activities. In Figure 6 we observe that giving self-study a bigger penalty leads to less learners in self-study. This is indeed expected, since the bigger penalty results in more modules being preferred over self-study for learners, which leads to more possibilities to schedule a learner in an instruction activity.

In Figure 10 it is shown what splitting classrooms does for the schedule quality. As expected, it reduces the number of learners in self-study, as there can now be more activities scheduled of different modules. It comes at a risk of taking away larger activities, namely, those of size bigger than 16. However, this is only an issue for very small demand spreads and larger schools. In most cases it is already hard to schedule groups of those large sizes.

Finally, we investigate the effect of teacher qualification distribution. In Figure 8 it is shown that the difference between (1;0;0) and (0.5;0.5;0) is negligible. Therefore, it is economically efficient to hire both first-degree and second-degree teachers. The introduction of third-degree teachers does degrade the schedule quality somewhat. It results in an increase of self-study learners of around 10%. We then varied the teacher qualifications by introducing a variable q. The teacher distribution can be written as $(1-q; 0; q)$. Experiments have been generated varying q in $(0, 0.1, ..., 0.9, 1)$. Also each of the demand spreads and school sizes are taken into account. The self study parameter ω is set to 50%, δ^- to 5, δ^+ to 30 and the classrooms are not split. This results in 132 experiments, which we solved with Grey Wolf given 60 seconds of runtime. Figure 11 shows the effect of different values of q on the solution quality, for different school sizes. We can see that for values of 0.5 and lower, the solution quality stays approximately the same. At higher values, the solution quality quickly degrades, and at a value of 1 all learners are in self-study. These results are seen for all school sizes.

As expected all these results confirm the results found by Wouda et al. (2023). Since we used the same data this was to be expected. The only real difference is the solution quality. The Grey Wolf results across the board have slightly more learners in self-study. This does not affect the impact changing the parameters have on the solution quality however.

6 Conclusion

In personalized learning, scheduling is one of the main challenges. Wouda et al. (2023) have proposed both a problem, the HLAPP, and a metaheuristic solution method, ALNS, for re-scheduling purposes in personalized learning. We have extended their research by introducing a new solution method for this problem, in the form of the Grey Wolf algorithm. We have found that Grey Wolf outperforms ALNS in 429 out of 576 instances of our dataset. It also does this in a comparable amount of time, namely 60 seconds. In general, the ILP formulation of the HLAPP problem, given 10 minutes of runtime, yields the best results. However, this is not a fair comparison. The ILP has a complexity that results in a high memory requirement to even start solving. In many cases the common 16gb of memory is not enough to solve. That is where ALNS and Grey Wolf come in. They do not require large amounts of memory, or a very powerful processor. When comparing these two, Grey Wolf clearly performs better. Both on average, and in most instances. The only exception is experiments with a demand spread of zero, which in practice is not realistic. Our conclusion is therefore that Grey Wolf is a better solution method for the HLAPP problem, especially for real-world applications.

In this real-world, schools might find it beneficial to make certain policy decisions to improve schedule quality. We show that it is beneficial to split classrooms, resulting in double the classrooms of half the size. This can be achieved with the installations of partition walls in regular classrooms. Also, it is shown that having a mix of first-degree and second-degree teachers is economically efficient. Schools have the choice to employ a small amount of third-degree teachers, but this comes at a small cost of schedule quality. When classrooms are not split, there is more room for third-degree teachers without affecting schedule quality.

There are some limitations to our research. The metaheuristic and Grey Wolf Algorithm make obtaining quality schedules in short amount of times possible. However, the HLAPP problem itself is not quite ready for practical use. The problem is proposed for re-scheduling purposes. Meaning, there already exists a more long term schedule but due to short term shocks (for example a teacher becoming unavailable), certain activities have to be rescheduled. In practice, it may be very important to make as little changes as possible to the long-term schedule. The HLAPP problem makes an entirely new schedule for the single hour, which might result in many different assignments and activities. We therefore recommend extending our research to take into account a long-term schedule. We suggest adding a soft-constraint to the objective function, which takes into account the number of changes made to the long-term schedule. This could however make the model more complicated to solve.

We also recommend further research into the Grey Wolf algorithm. A clear improvement that could be made is the addition of hyperparameter optimization. Also, a different way of selecting mutation operators might improve the algorithm. For example, a roulette wheel or a neural network could be used to select mutation operators. Another thing that could improve the algorithm is the addition of more mutation operators. A swap move is often used in literature, where two assignments are swapped. Another improvement that could be made is extending the crossover operator to take into account the learner assignments from the alpha, beta and delta wolfs, where they did not have the same assignment. By simply ignoring those assignments, setting these learners to unassigned and using greedy insert, a lot of information is lost. Finally, it may be worthwhile to look into the addition of a local search procedure.

References

Al-Betar, M. A., Awadallah, M. A., Faris, H., Aljarah, I. & Hammouri, A. I. (2018). Natural selection methods for grey wolf optimizer. *Expert Systems with Applications*, *113*, 481-498. doi: https://doi.org/10.1016/j.eswa.2018.07.022

Almufti, S., Ahmad, H., Marqas, R. & Asaad, R. (2021, 08). Grey wolf optimizer: Overview, modifications and applications. , *44*, 1-13. doi: 10.5281/zenodo.5195644

Aslan, A., Bakir, I. & Vis, I. F. (2020). A dynamic thompson sampling hyper-heuristic framework for learning activity planning in personalized learning. *European Journal of Operational Research*, *286*(2), 673-688. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0377221720302526` doi: https://doi.org/10.1016/j.ejor.2020.03.038

Asratian, A. & Werra, D. (2002, 02). A generalized class-teacher model for some timetabling problems. *European Journal of Operational Research*, *143*, 531-542.

Eshelman, L. J. & Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. In L. D. WHITLEY (Ed.), *Foundations of genetic algorithms* (Vol. 2, p. 187-202). Elsevier.

Kannan, A., van den Berg, G. & Kuo, A. (2012, October). iSchedule to Personalize Learning. *Interfaces*, *42*(5), 437-448.

Kiefer, A., Hartl, R. & Schnell, A. (2017, 05). Adaptive large neighborhood search for the curriculum-based course timetabling problem. *Annals of Operations Research*, *252*. doi: 10.1007/s10479-016-2151-2

Kristiansen, S., Sørensen, M. & Stidsen, T. R. (2011, December). Elective course planning. *European Journal of Operational Research*, *215*(3), 713-720.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., ... Hutter, F. (2022). Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, *23*(54), 1–9. Retrieved from `http://jmlr.org/papers/v23/21-0888.html`

Mirjalili, S., Mirjalili, S. M. & Lewis, A. (2014). Grey wolf optimizer. *Advances in engineering software*, *69*, 46–61.

Nadimi-Shahraki, M. H., Taghian, S. & Mirjalili, S. (2021). An improved grey wolf optimizer for solving engineering problems. *Expert Systems with Applications*, *166*, 113917.

Ngoo, C. M., Goh, S. L. & Kendall, G. (2022, 10). An improved discrete grey wolf optimizer for the nurse rostering problem.
doi: 10.21203/rs.3.rs-2164519/v1

Ngoo, C. M., Goh, S. L., Sabar, N., Sze, S., Abdullah, S. & Kendall, G. (2022, 01). A survey of the nurse rostering solution methodologies: The state-of-the-art and emerging trends. *IEEE Access*, *10*, 1-1.

Pillay, N. (2013, 02). A survey of school timetabling research. *Annals of Operations Research*, *218*. doi: 10.1007/s10479-013-1321-8

Pisinger, D. & Ropke, S. (2010, 09). Large neighborhood search. In (p. 399-419).

Santiago-Mozos, R., Salcedo-Sanz, S., DePrado-Cumplido, M. & Bousoño-Calzón, C. (2005, 07). A two-phase heuristic evolutionary algorithm for personalizing course timetables: A case study in a spanish university. *Computers Operations Research*, *32*, 1761-1776. doi: 10.1016/j.cor.2003.11.030

Veenstra, M. & Vis, I. (2016, 03). School timetabling problem under disturbances. *Computers Industrial Engineering*, *95*. doi: 10.1016/j.cie.2016.02.011

Wouda, N. A., Aslan, A. & Vis, I. F. (2023). An adaptive large neighbourhood search metaheuristic for hourly learning activity planning in personalised learning. *Computers Operations Research*, *151*, 106089.

Wouda, N. A. & Lan, L. (2023). ALNS: a Python implementation of the adaptive large neighbourhood search metaheuristic. *Journal of Open Source Software*, *8*(81), 5028. Retrieved from `https://doi.org/10.21105/joss.05028` doi: 10.21105/joss.05028

A Figures of effects of parameters

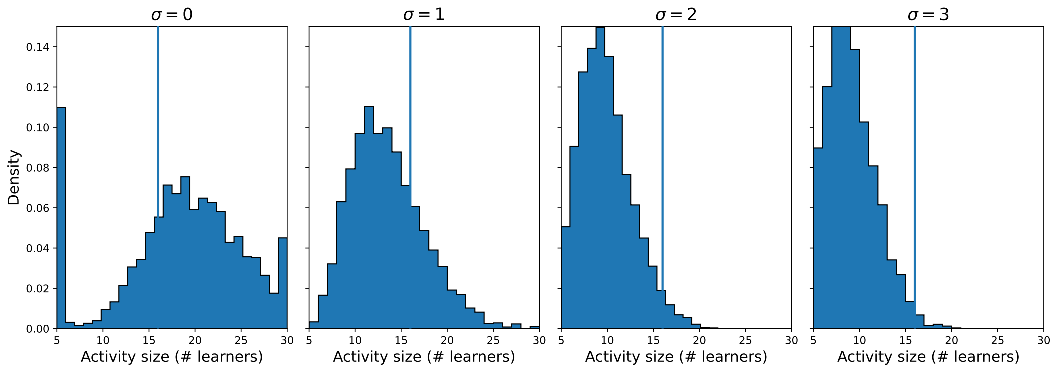

Figure (5) Histograms of the activity sizes for each demand spread

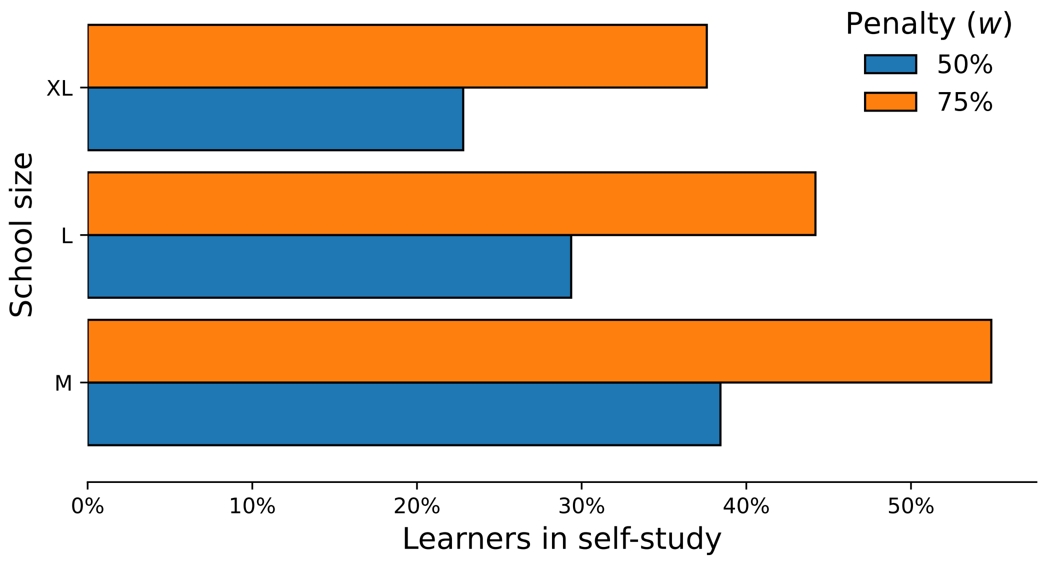

Figure (6) Solution quality for different penalty's

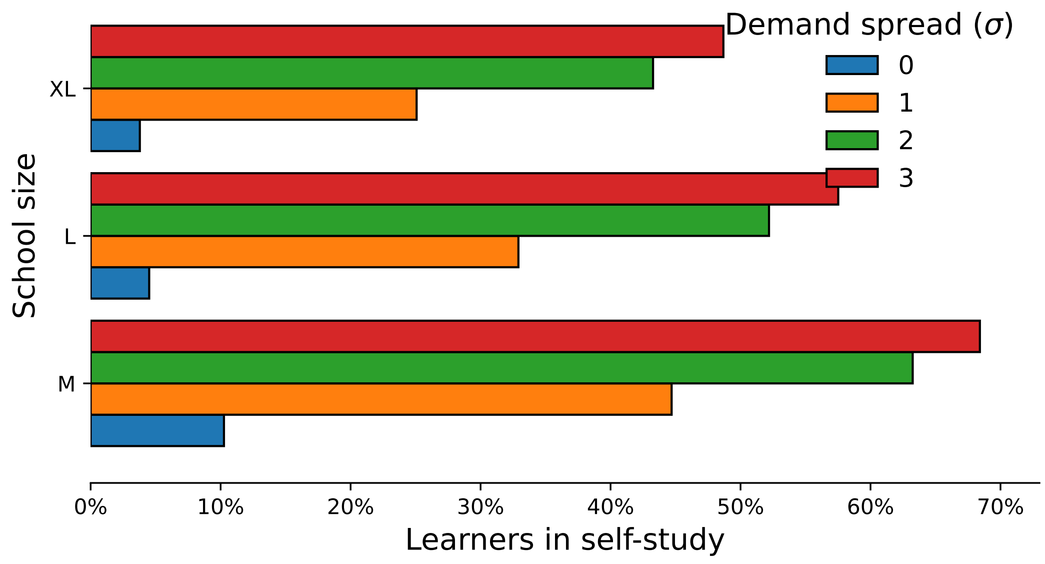

Figure (7) Solution quality for different demand spreads

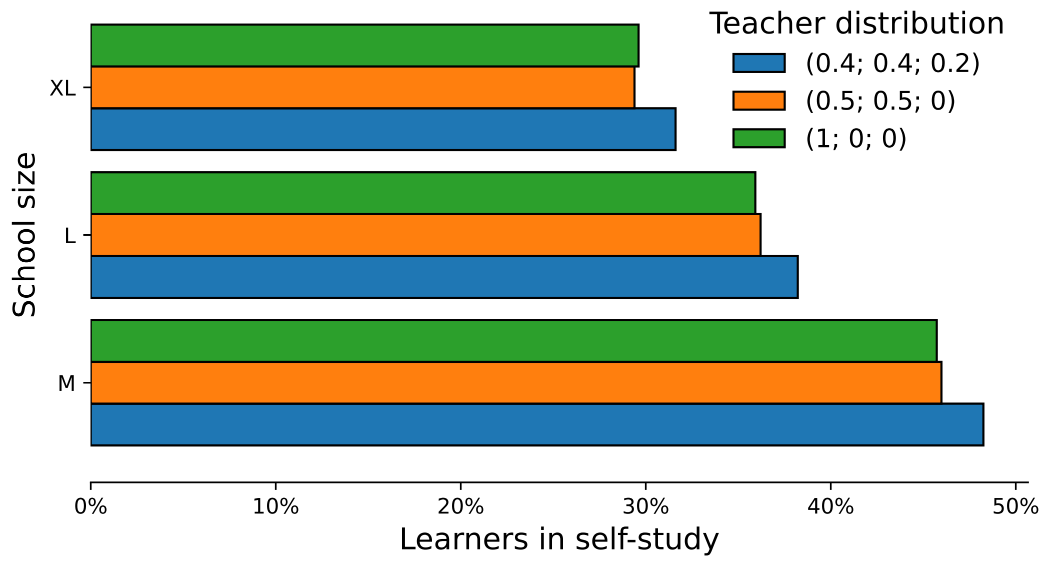

Figure (8) Solution quality for different teacher distributions

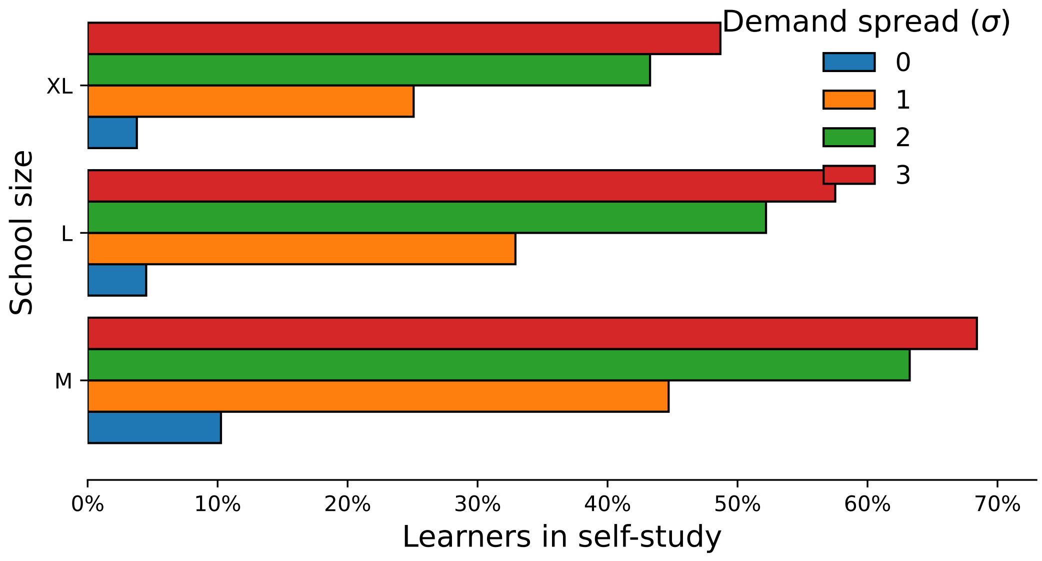

Figure (9) Solution quality for different demand spreads

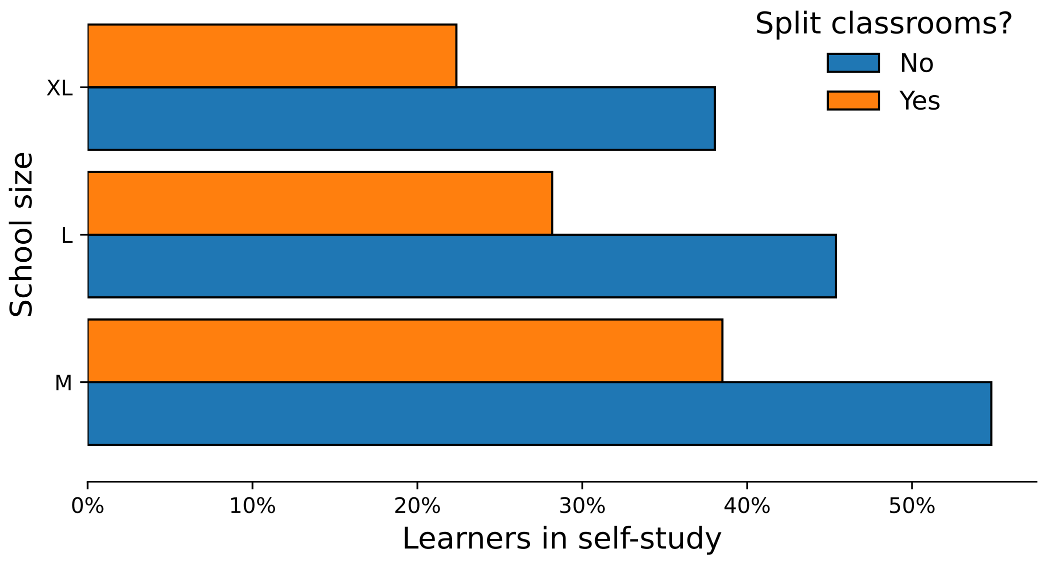

Figure (10) Solution quality for the classroom split

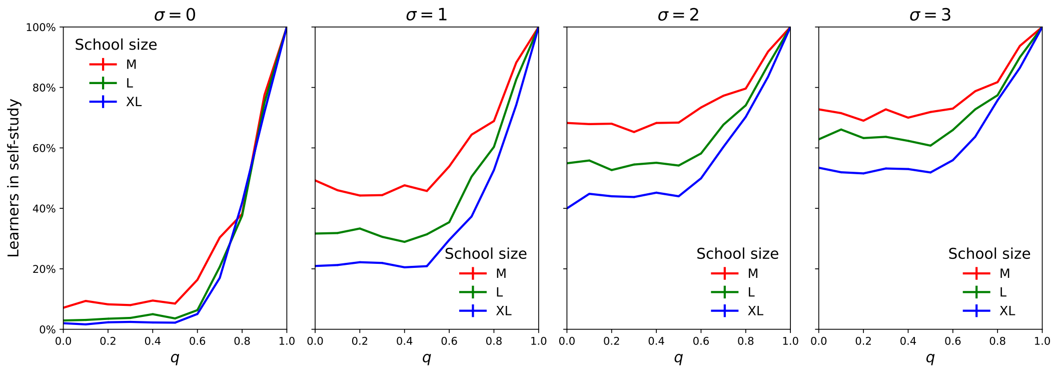

Figure (11) Solution quality for many different teacher distributions

B Programming code

B.1 Inside src

analyse.py: Code that creates tables that report all important findings for an experiment for a specific solution method.
constants.py The set parameters used by ALNS (unchanged)
greywolf.py The main algorithm of Grey Wolf, it calls the mutation and crossover operators. Outputs a best solution.
heuristic.py The main heuristic file, it calls the alns package.
ilp.py The maian ILP file, it creates the problem, sets constraints and calls on gurobi to solve.
make_experiments.py It creates the experiment instances (unused).
results.py The facts and figures of the extension are reported from here.
results_paramater_effects.py Here the figures are created for the replication part, where the parameter effects are measured.
runAnalyse.py A simple script that loops through the experiments and runs analyse.py for all of them.
runGreywolf.py A simple script that loops through experiments and instances and runs greywolf.py for all of them.
runHeuristic.py A simple script that loops through experiments and instances and runs the heuristic for all of them.
runIlp.py A simple script that loops through experiments and instances and runs the ilp.py for all of them.
runValidator.py A simple script that loops through experiments and instances and runs validor.py for all of them.
tune.py A script that runs the hyperparameter optimization (unused).
validator.py A script that checks if the obtained solution for a or multiple solution methods is valid.

B.2 Inside classes

For all of these classes no meaningful changes can be reported.
Activity.py
Classroom.py
Learner.py
Module.py
Problem.py
Result.py
Solution.py
Teacher.py

B.3 Inside crossover_operators

discrete_blend_crossover.py This is the crossover operator as explained in the methodology. It calls upon greedy_insert to reassign unassigned learners.

B.4 Inside destroy operators

For all of these classes no meaningful changes can be reported.
random_activities.py
random_learners.py
regret_learners.py
smalles_activities.py

B.5 Inside functions

For all of these classes no meaningful changes can be reported.
initial_solution.py
learners_to_remove.py
pairwise.py
problem.py

B.6 Inside local_search

For this class no meaningful changes can be reported.
reinsert_learner.py

B.7 Inside mutation_operators

activity_reassignment.py This class removes random activities from the solution, then applies break_out to create new activities and finally applies greedy_insert to assign the last learners to the new solution.
greedy_insert.py This class greedily inserts unassigned learners to modules. This is not a mutation operator on its own.
learner_reassignment.py This class removes random learners from their assignments and removes activities if they get too small. It then applies break_out to create new activities. Finally greedy_insert is run to assign the final unassigned learners.

B.8 Inside repair_operators

For all of these classes no meaningful changes can be reported.
break_out.py
greedy_insert.py

B.9 Inside rules

These classes represent the restrictions to check. No meaningful changes can be reported.

activity_size.py

classrooms_to_modules.py

classrooms_to_teacers.py

learner_preferences.py

learners_to_classrooms.py

learners_to_modules.py

learners_to_teachers.py

module_classroom_room_type.py

teacher_module_qualifications.py

teachers_to_classrooms.py

teachers_to_modules.py

B.10 Data

The used data and results can be found in the map 'experiments' inside 'src'.