Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Analytics and Operations Research in Logistics

The Daily Learning Activity Planning Problem: Enhancing Student-Centered Learning Through Day-Based Dynamic Scheduling

Oscar de Groot
Student number: 544492

Abstract

This paper presents a refinement of the hourly learning activity planning problem for personalized learning schools, called the daily learning activity planning problem. This new problem extends the original one by allowing for full-day scheduling of learners for multiple hours, which enables the optimization of learner demand satisfaction over an entire day. In addition, this model introduces a new restriction, which allows for the incorporation of a decrease in learner-module efficiency when studying one particular module more than once in a school day. To solve this problem, an adaptive large neighborhood search metaheuristic is used. The proposed model and heuristic are applied to the scheduling of learner-centered schools, which focus on personalized learning, but face challenges in short-term scheduling due to dynamic learner preferences. The results are very promising. Across different experiment settings, the heuristic can solve the problem 8.50% from optimality on average. This study contributes to the literature by introducing a new problem definition and solution approach for personalized learning schools, which can help improve short-term scheduling and learner satisfaction. Heuristic solutions can be improved further in future research by creating a more advanced algorithm or by applying stronger hardware.

Supervisor:	J.H. Zhu
Second assessor:	Prof. Dr. D. Huisman
Date final version:	2nd July 2023

Contents

1 Introduction

Traditionally, in the Netherlands, the most standard system for secondary education consists of constructing a regular, weekly timetable that assigns learners to classrooms, so that they can learn the module planned for that hour from their assigned teacher. However, over the past years, education systems that are more learner-, rather than class-oriented have been uprising (McCarthy, Liu & Schauer, 2020). Take for example *Montessori education*. Montessori education is an educational approach developed by Maria Montessori that emphasizes self-directed learning, independence, and respect for the child's individuality. This approach entails creating a well-equipped setting with interactive resources and cultivating a nurturing ambiance that enables children to independently explore, acquire knowledge at their preferred pace, and enhance their cognitive, social, and emotional abilities (Foundation, 2022; Hiles, 2018). This method has proven a degree of effectiveness over the past century (Marshall, 2017; Faryadi, 2017), giving rise to the question of what other education types carry potential.

A more novel approach to education is the newly introduced *Kunskapsskolan*, which was first introduced in Sweden in 2000 and opened its first school in The Netherlands in 2014 (Kunskapsskolan, 2023). This school system focuses on facilitating personalized learning (PL) (Eiken, 2011). In PL, the content a student learns in a given hour is more aligned with their individual needs. The learner has more autonomy concerning what module they will learn, since the school will try to adhere to the needs of the individual, rather than assigning a more generalized schedule to a group of learners, like a class. The novel Kunskapsskolan is already being implemented by more than a hundred schools around the world and is even regarded as an important instrument in facilitating the transformation of education by the European Commission (Commission & Centre, 2019).

However, the more personalized scheduling style employed by Kunskapsskolan does pose challenges in terms of school timetabling. The dynamic, short-term-focused learner preferences applied by this school type are poorly adhered to by classical scheduling methods. This is why this schooling type calls for appropriate short-term scheduling methods.

This paper introduces the daily learning activity planning problem (DLAPP), extending the hourly learning activity planning problem (HLAPP) of PL schools introduced by Wouda, Aslan and Vis (2023), and analyses its effectiveness. This novel model supplements a longer calendar by allowing one to make a full new schedule for a (remainder of a) school day consisting of a predetermined number of hours. In contrast to the HLAPP, this refined model allows for scheduling a sequence of hours. The advantage of this feature is that one can optimize learner satisfaction throughout the day or, in the event of a disruption, adjust the schedule for the remaining time period based on current learner demands for activities. This adds value in cases where a classroom or teacher suddenly might become unavailable, and a new schedule must rapidly be constructed. In case it is known that a teacher will not be present on a certain day, another added benefit of this scheduling method is that a learner will instantly know what their calendar will be for the entire day, which allows for better course preparation. To acquire feasible solving times, the metaheuristic proposed for the HLAPP, which can near-optimally solve the problem rapidly by using a complex algorithm that incorporates randomness, is adapted to properly solve the extended problem.

To adequately account for some real-life properties of a school day, new restrictions are applied. The model accounts for the feature concerning decreasing learner efficiency when having to study one particular module for multiple hours in a day. This is caused by learners having decreased focus when facing a monotonous schedule and by the fact that learners might not be able to prepare for the next session on the same subject properly when no time is given for that (Brown & Saks, 1987). By incorporating this feature into the model, it is ensured that when learner preferences are given as a constant throughout the entire day, the learners will not simply be assigned to their single most preferred topic throughout the entire day. Instead, they will learn different topics, facilitating a productive, varied study schedule.

Since this problem, just like the HLAPP, might require very swift solving times in case a teacher suddenly becomes unavailable for the day, a heuristic is needed. This paper introduces an adaptive large neighborhood search (ALNS) metaheuristic similar to the one proposed by Wouda et al. (2023).

This renewed heuristic iteratively destroys and repairs parts of an initial set of assignments that forms a feasible solution. In every iteration, a single hour is selected from which the assignments may be adjusted. It can destroy and repair the solution by destroying and creating new activities, or by removing, reinserting, or reallocating singular learners. Some of the operations, especially the ones that reassign learners to activities or create new ones, keep into account the level of overall utility gained if that change were to be made. This allows for increased objective gain in every iteration.

To compute what moves can create the largest increase in utility, the current number of times each learner takes each module throughout all hours has to be tracked at all times. These quantities namely are required to obtain the current utilities the learners gain from each module throughout the day, taking into account the imposed penalty terms if learners were to face a monotonous schedule. In addition to the current utilities of each learner, the hypothetical utilities, if the quantities were to either decrease or increase, are also tracked. These values are required when computing overall gain or loss made if a certain change in the schedule were to be made.

The main question this research targets to answer is: *How potent is the DLAPP for school timetabling, and how well does the adapted metaheuristic perform when solving this problem?* Both run-times and gaps to optimality are decisive in the analysis of the performance.

This research uncovers that the reduced number of iterations that can be applied by the metaheuristic, in combination with the enlarged model complexity cause the heuristic to have an average gap of 8.50% when compared to the optimal solution. The model however stills shows promising literary value. Since the DLAPP framework has been laid, stronger hardware and a more advanced heuristic solution framework might prove to deliver lower optimality gaps when required to solve within a few minutes. More parameter settings can also be explored to further analyze the performance of the heuristic.

The contents of this paper will be structured as follows. Section 2 will introduce the parameters and variables used in the model. Section 3 introduces the formulation of the DLAPP and Section 4 will explain the workings of the ALNS metaheuristic used in this research. In Section 5, the HLAPP and the performance of its original metaheuristic are analyzed. Then, in Section

6, the experiments used to evaluate the performance of the metaheuristic applied to the DLAPP will be discussed, as well as the tuning procedure and the results. Section 7 discusses the caveats that can be taken from the results of this paper and lastly, in Section 8, the shortcomings of this paper and possibilities for further research will be addressed.

2 Problem definition

To solve the problem, the goal is to create a set of *activities* for every hour $h \in H$, which learners can be assigned to. These activities each consist of a group of learners, a teacher, a classroom, and the module that is taught in it. The available set of resources that will be distributed across these activities is a set of learners $l \in L$ who have demand across a set of modules $m \in M$. The modules should be taught by a set of teachers $t \in T$ across a set of classrooms $c \in C$. To assign these resources over the set of hours, the most up-to-date learner demands for modules ($D_{lm} \in \mathbb{R}_{\geq 0}$) will be used. To denote a learner's l ineligibility for module m (across all hours), we set $D_{lm} = 0$. The target is to assign each learner l to modules m for each hour h such that the learners are met most in their needs across the day. A learner-module assignment for a certain hour is denoted with the decision variable $y_{lmh} \in \{0, 1\}$.

In contrast to the HLAPP, the 'utility' a learner gains across the day does not simply consist of the preference he or she has for a module, multiplied by the number of times the module is taken. Since it is assumed that learner efficiency decreases when taking certain modules more than once a day, a selected penalty can be enforced. In the case that $|H| > 1$, when learner l takes module m more than one time in that schedule, a monotonicity penalty measure Λ ($0 \leq \Lambda \leq \frac{1}{|H|-1}$) is applied to that learner-module combination. Λ represents the percentage points of utility lost for every number above one concerning the number of times the learner takes the module that day. The percentage of utility left that learner l gains from module m is denoted by $p_{lm} \in \mathbb{R}$. Note that this level of productivity p_{lm} does not need delimitation, since the nature of the model will always try to give this variable a value that is as high as possible, and will be delimited with a maximum for every learner module combination. The minimum value of Λ is set at 0 since a negative value would imply increased productivity in a monotonous schedule. This case is excluded from this research. In the case that $|H| = 1$, the value set for Λ is arbitrary, since monotonicity penalties will never be enforced.

Lemmas 1 and 2 state the delimitations on Λ, which are proven in Appendix A.

Lemma 1. *Let Λ be the monotonicity penalty measure applied to the utility learner l gains from module m in a day with $|H| > 1$ hours in it. If a learner l should gain positive utility from module m by taking it any number of times that day, $\Lambda \leq \frac{1}{|H|-1}$.*

Lemma 2. *Let Λ be the monotonicity penalty measure applied to the utility learner l gains from module m in a day with $|H| > 1$ hours in it. If a learner l should always gain more utility by partaking in module m one more time that day, $\Lambda \leq \frac{1}{2|H|-1}$.*

The first, least tight restriction from Lemma 1 should be seen as essential, since gaining negative utility from partaking in activities is unrealistic, and also not applied in the HLAPP. Violating the tighter restriction from Lemma 2 delivers a more realistic scenario, but is still not recommended. If learner l already partakes in module m a large number of times in a day, the

Table 1: Definitions of the input and decision variables used.

Notation	Definition
$l \in L$	Set of learners.
$c \in C$	Set of classrooms.
$t \in T$	Set of teachers.
$m \in M$	Set of modules, including m_s, representing the self-study module.
$h \in H$	Set of hours that will take place during (the remainder of) the school day.
$\Lambda \in \begin{cases} [0, \frac{1}{\lvert H \rvert - 1}] & \text{if } \lvert H \rvert > 1 \\ \mathbb{R} & \text{if } \lvert H \rvert = 1 \end{cases}$	Monotonicity penalty applied for every number of times a module is taken more than one time in a day.
$D \in \mathbb{R}_{\geq 0}^{[L \rvert \times \lvert M \rvert}$	The demand matrix. A value D_{lm} gives the demand of learner $l \in L$ for module $m \in M$. Demand is 0 when the learner is not eligible for the module.
$Q^T \in \{0,1\}^{\lvert T \rvert \times \lvert M \rvert}$	Teacher qualification matrix. $Q^T_{\text{tm}} = 1$ if teacher $t \in T$ can teach module $m \in M$, 0 otherwise.
$Q^C \in \{0,1\}^{\lvert C \rvert \times \lvert M \rvert}$	Classroom qualification matrix. $Q^C_{cm} = 1$ if classroom $c \in C$ can host an activity for module $m \in M$, 0 otherwise.
$\omega \in [0,1]$	Self-study penalty applied to the m_S module for learner l to balance instruction and self-study activities.
$\delta^- \in \mathbb{Z}_{\geq 0}$	Minimum activity size for all modules.
$\delta^+ \in \mathbb{Z}_{\geq 0}$	Maximum instruction activity size for module m.
$N_c \in \mathbb{N}$	The capacity of classroom $c \in C$.
$x_{mcth} \in \{0,1\}$	Decision that is 1 if an activity with module $m \in M$ is scheduled in classroom $c \in C$, taught by teacher $t \in T$, 0 otherwise.
$y_{lmh} \in \{0,1\}$	Decision that is 1 if learner $l \in L$ is assigned to module $m \in M$, 0 otherwise.
$p_{lm} \in \mathbb{R}$	Variable indicating productivity which decreases as a learner $l \in L$ is assigned to module $m \in M$ more often.

added benefit of partaking in an activity where module m is taught might deliver close to 0 marginal utility, but it can be seen as a far fetch to claim it reduces summed up utility gained from the module over the entire day.

To denote the allocation of a module m being taught in classroom c given by a teacher t in hour h, the decision variable $x_{mcth} \in \{0,1\}$ is applied. When this decision variable is equal to 1, we call this an activity. In the same fashion as Wouda et al.(2023), two types of activities are available. First off, a regular activity in which a module is being taught by a qualified teacher, making it so that the entire class works on the same module, with their personal demand D_{lm} times their level of productivity p_{lm} denoting the benefit they receive from it. In the PL style that Kunskapsskolan schools apply, however, self-study activities are also available. These modules can be given by any teacher, who will function as a supervisor. This activity will be denoted by $m_S \in M$. During the session, learners will have a free choice concerning which module they will be studying. Since self-study sessions are not supplemented by intensive education from qualified teachers, a penalty ω $(0 \leq \omega \leq 1)$is applied to each learner. This penalty denotes the fraction of utility a learner l gains from learning their module of choice during the self-study session. Since demands are clearly defined, it is assumed that learner will be learning their most preferred module. This leads to a demand level for m_S given by $D_{lm_S} = \omega \max_{m \in M \setminus m_S} D_{lm}$ for the self-study module. Note that p_{lm} is also applied to the self-study module.

When allocating activities across the decision variables x_{mcth}, it should be accounted for that a certain teacher t might not be qualified to teach module m. To denote whether a teacher t is qualified for module m, a binary qualification matrix $Q^T_{tm} \in \{0,1\}$ is put into place. Note that the fact that all teachers are available to supervise a self-study activity is denoted by $Q^T_{tm_S} = 1 \quad \forall t \in T$. In the same fashion, a classroom c might not be suitable for teaching a

certain module m. This is tracked by the binary qualification matrix $Q^C_{cm} \in \{0,1\}$.

Since a classroom c has a maximum capacity for learners, a capacity $N_c \in \mathbb{N}$ is assigned to the set of classrooms $c \in C$. Additionally, the DLAPP imposes restrictions concerning learner quantities for the modules themselves. Since very small activities might lead to unnecessary use of classrooms or inefficient teacher use, a minimum activity size $\delta^- \in \mathbb{N}$ is imposed. The research by Wouda et al.(2023) also imposes a maximum activity size $\delta^+ \in \mathbb{N}$ in a similar fashion. This measure limits the maximum number of learners that can be in an instruction activity, meaning it is not applied to self-study activities. This will be applied to the extended model as well.

An overview of the definitions used in the model can be seen in Table 1.

3 Model

The DLAPP is constructed as follows. The target is to optimize the objective

$$\max_{x,y,p} \sum_{l \in L} \sum_{m \in M} D_{lm} p_{lm} \sum_{h \in H} y_{lmh} \tag{1}$$

subject to

$$\sum_{l \in L} y_{lmh} \leq \sum_{c \in C} \min\left(\delta^+, N_c\right) \sum_{t \in T} x_{mcth} \quad \forall m \in M \setminus \{m_S\}, \forall h \in H \tag{2}$$

$$\sum_{l \in L} y_{lm_S h} \leq \sum_{c \in C} N_c \sum_{t \in T} x_{m_S cth}, \quad \forall h \in H \tag{3}$$

$$\sum_{l \in L} y_{lmh} \geq \delta^- \sum_{c \in C} \sum_{t \in T} x_{mcth} \quad \forall m \in M, \forall h \in H \tag{4}$$

$$\sum_{m \in M} y_{lmh} = 1 \quad \forall l \in L, \forall h \in H \tag{5}$$

$$\sum_{m \in M} \sum_{c \in C} x_{mcth} \leq 1 \quad \forall t \in T, \forall h \in H \tag{6}$$

$$\sum_{m \in M} \sum_{t \in T} x_{mcth} \leq 1 \quad \forall c \in C, \forall h \in H \tag{7}$$

$$\sum_{c \in C} x_{mcth} \leq Q^T_{tm} \quad \forall t \in T, \forall m \in M, \forall h \in H \tag{8}$$

$$\sum_{t \in T} x_{mcth} \leq Q^C_{cm} \quad \forall c \in C, \forall m \in M, \forall h \in H \tag{9}$$

$$y_{lmh} \leq \mathbf{1}_{D_{lm} > 0} \quad \forall l \in L, \forall m \in M, \forall h \in H \tag{10}$$

$$p_{lm} \leq 1 + (1 - \sum_{h \in H} y_{lmh})\Lambda \quad \forall l \in L, \forall m \in M, \tag{11}$$

$$x_{mcth} \in \{0,1\} \quad \forall m \in M, \forall c \in C, \forall t \in T, \forall h \in H \tag{12}$$

$$y_{lmh} \in \{0,1\} \quad \forall l \in L, \forall m \in M, \forall h \in H \tag{13}$$

$$p_{lm} \in \mathbb{R} \quad \forall l \in L, \forall m \in M \tag{14}$$

Constraints 2 ensure that there are no more learners assigned to a certain module for a particular hour than the assigned number of corresponding class/module combinations can handle. Constraints 3 ensure the same thing but for the self-study module. Constraints 4 ensure that there are enough learners so that all corresponding class/module combinations can fulfill the minimum learner requirements. Constraints 5 guarantee that all learners are assigned to some module every single hour. Constraints 6 prevent teachers from being assigned to more than one module in a single hour. Constraints 7 do the same thing but for classrooms instead. Constraints 8 ensure that teachers can only give modules that they are qualified to give for every hour. Constraints 9 do the same thing for classrooms. Constraints 10 make sure that learners with 0 demand for a module can never partake in an activity where that module is taught. This is done using an indicator function that is only equal to one if demand is not 0. Constraints 11 enforce the penalty measure for monotonous study schedules. Lastly, Constraints 12 and 13 restrict the decision variables to be binary and Constraint 14 indicates that productivity based on monotonicity can be any real value.

The definition of the model is very similar to the one of the HLAPP. The differences are that in the model for the HLAPP, p_{lm} and Constraints 11 do not exist. In addition to that, x_{mcth} and y_{lmh} are reduced to x_{mct} and y_{lm}, and all of the remaining constraints do not have to be applied for every $h \in H$, but just for the other sets of recourses remaining, if any.

The goal of a constraint from the set of Constraints 11 is to impose a penalty term that will be incurred for module m learned by learner l on that day. This value is equal to 1 if a learner takes a module 1 time that day, and decreases by Λ for every additional time that module is taken that day. Note that p_{lm} is allowed to rise up to $1+\Lambda$ if learner l takes module m zero hours that day. This 'increased productivity' however is never applied to the objective function, since y_{lmh} will always be 0 whenever this is the case. The result of solving the model is an optimal solution for the decision variables x_{mcth}, y_{lmh}, and p_{lm}. Note that p_{lm} does not contribute to the actual construction of the assignments, since it solely functions as a penalizing variable when solving the model.

Since similarly to the HLAPP, x_{mcth} grows three-dimensionally as school size increases, this problem is also NP-hard. Note that the size of set H does not grow simultaneously as school size increases. The number of hours $|H|$ included in the model can range from one up to any number of hours desired. This quantity will probably rarely be higher than ten, since schools do not regularly have more available timeslots for activities than that amount. The problem however rapidly grows even further when the number of hours is increased. Since the number of restrictions grows approximately linear in the number of hours added on top of the regular one-hour scheduling problem, and interdependence becomes more and more complex through Constraints 11 as hours increase, it becomes very computationally demanding even for small schools when trying to plan for more than a couple of hours. This is why the DLAPP calls for a well-performing heuristic.

4 Metaheuristic description

To solve the problem in a feasible time span, allowing schools to quickly renew their schedule for the rest of the day, an adaptive large neighborhood search (ALNS) metaheuristic is proposed.

The ALNS procedure was originally introduced by Ropke and Pisinger(2006). To generally describe the ALNS algorithm, Algorithm 1 is given. In this section, a description is provided concerning the workings of this heuristic.

Algorithm 1: Adaptive large neighborhood search.

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

As can be seen in the algorithm, the process is started by creating an initial, feasible solution, in which every learner is assigned to a module for every hour. Since the algorithm will go through many iterations, the initial solution does not need to be very good. The algorithm then loops for a set number of iterations. In every iteration, firstly, a single hour $h \in H$ is selected which will be worked on. In the rest of the iteration, it is only allowed to adjust assignments in that hour. After selecting the hour, a destroy and repair operator are selected out of set O_D and O_R respectively. These operators transform the current solution s into a new candidate solution s^c. After evaluating this outcome, the operator selection mechanism is adjusted.

In this section, there will be an explanation of how the ALNS procedure computes the initial solution. Then, the set of destroy and repair operators will be discussed. Afterward, the acceptance criterion, which determines whether the candidate solution will be used, will be explained. The local search mechanism that is applied when a new solution is adapted will then be explained. Lastly, the part of the metaheuristic that updates the weights used for selecting the destroy and repair operators will be discussed. Note that this last algorithm is what makes the entirety of the process *adaptive*.

4.1 Initial solution

To construct the initial solution, all learners are assigned to the self-study activity in every single hour. This solution will be inefficient for the DLAPP, especially when high values of Λ are applied. The solution is constructed by selecting a random classroom c and a random teacher t that can accommodate a self-study module and then allocating N_c learners into the module. This process is repeated until all learners are assigned.

4.2 Destroy operators

A set of four destroy operators is used for removing learners from the current solution throughout the iterations. They each remove a proportion of all learners $d > 0$ from their current

assignments. Each operator starts by randomly selecting the hour h which will be worked on.

4.2.1 Random activity removal

This operator removes random activities in the selected hour h until at least d learners have been removed. All learners that were in the removed activity are marked as unassigned, just like the classroom and teacher corresponding to it.

4.2.2 Smallest activity removal

This operator functions in nearly an identical way as random activity removal, however, it instead specifically removes the smallest activities. This makes it so that classrooms and teachers which are likely to be used inefficiently are freed up.

4.2.3 Random learner removal

This operator removes random learners from their activities in the selected hour h until at least d learners have been removed. A learner is only removed from their activity if that activity remains feasible, namely, when δ^- is respected.

4.2.4 Worst regret learner removal

This operator is the first one to work differently from the operators used in Wouda et al.(2023). The idea behind this operator is that it removes learners who have the best possibility of improving their utility by being unassigned from the module they are currently assigned to in the hour which is being adjusted. The 'regret', which denotes this possibility of improvement, is calculated for each learner $l \in L$. The regret of a learner is denoted as:

$$r_{lm} = \max_{m' \in M} \{U_{lm'}\} - U_{lm_{curr}}, \tag{15}$$

where

$$U_{lm} = \sum_{m \in M} D_{lm} p_{lm} \sum_{h \in H} y_{lmh}. \tag{16}$$

In Equations 15 and 16, U_{lm} denotes the total utility learner l gains across the entire problem, if they are in module m in the hour currently being worked on. This equates to $U_{lm_{curr}}$ if learner l is currently assigned to module $m_{curr} \in M$. On the other hand, $U_{lm'}$ denotes the potential total utility that learner l can have across the problem if its module in the hour being worked on were to switch to module m'. Note that the value of the objective function is equal to the sum of all utilities in a solution, which equates to $\sum_{l \in L} U_{lm}$.

If r_{lm} is high for learner l, this is considered worse. This is why the regrets are stored in decreasing order, whilst remembering the corresponding learner. d learners are then selected by using a skewed distribution, which favors selecting learners with large regrets over learners with small ones. The distribution is decreasing triangular for the first d values, and flat thereafter. This means it is uniform for all learners with index $\{d+1, ..., |L|\}$. The chance that a learner l

with its cost at index $\{1, ..., |L|\}$ is then selected has probability

$$\Pr(\text{select } l) = \begin{cases} \frac{d-j+1}{\sum_{i=1}^{d} i+|L|-j} & \text{if } j \leq d, \\ \frac{1}{\sum_{i=1}^{d} i+|L|-j} & \text{otherwise,} \end{cases} \tag{17}$$

of being selected. All of the selected learners are then removed from their current activities in the hour being worked on.

4.3 Repair operators

The repair operators make sure that all learners that are currently marked as unassigned are again assigned to an activity in the hour which is currently being worked on. Note that both of the two repair operators use overall utility U_{lm} from Equation 16 to determine assignments. This means they both account for other hours by incorporating possible changes in productivity if monotonicity were to alter because of an assignment.

4.3.1 Break-out activity

This operator attempts to create activities using the currently unassigned learners. Firstly, a set of modules is created, each having a set of corresponding unassigned learners, which have demand for the module. For each learner, the utility U_{lm} they would have by being assigned to this module has to be higher than U_{lm_S}, which is the utility they would have if they were to be assigned to the self-study module. This set of modules is then ordered by the level of overall utility $\sum_{l \in L} U_{lm}$ that can be gained by creating an activity for it and assigning its learners to it. Then, if a qualified classroom and teacher are available, the activity with the highest possible utility gain is created. In the setting of Dutch education, first-degree teachers exist who can each teach some set of modules. Additionally, second-degree teachers can only teach a subset of these modules, which are called second-degree modules. Instead of using a first-degree teacher for a second-degree module, the operator endeavors to select second-degree teachers specifically for those modules in this setting. In the same manner, when it comes to classrooms, the operator selects the smallest one that can accommodate the group of learners. If such a selection is not feasible, the operator opts for the largest available classroom instead.

Upon scheduling a new activity, the operator additionally attempts to assign self-study learners to the new activity provided if it offers a better alternative to their current self-study assignment. After creating a new activity, this process is repeated. If no new activities can be scheduled and some learners are still unassigned, the operator resorts to applying *greedy learner insert* to accommodate the remaining learners.

4.3.2 Greedy learner insert

This operator takes a random learner out of the set of unassigned ones and first opts to assign this learner to an existing feasible instruction activity within the hour being worked on. The operator attempts the insertion of a learner into an activity that causes the highest possible gain in U_{lm} for that learner. An insertion can only take place if the activity currently has less than

$\min\{\delta^+, N_c\}$ learners assigned to it, and the learner that is to be assigned to it has a non-zero demand for it.

In the absence of any viable instruction activities, the learner is allocated to an existing self-study activity. If that is not possible, a new self-study activity is established. In exceptional circumstances where even this proves unattainable, the instruction activity that contributes the least to the objective value is transformed into a self-study activity, and the learner is assigned accordingly. This process continues iteratively until all unassigned learners have been allocated to an activity.

4.4 Acceptance criterion

Whenever a candidate solution s^c is found, whether or not it is accepted is determined by a simulated annealing procedure. The reason for this is Santini, Ropke and Hvattum(2018) attaining excellent results using this procedure. The workings of the process entail that whenever a new solutionss^c is better than the current solution s, it is always accepted. Otherwise, whether or not it is accepted is determined by probability. This probability is dependent on the objectives of the solutions and is defined as

$$\Pr\left(\text{accept } s^c\right) = \exp\left\{\frac{f\left(s^c\right) - f(s)}{T}\right\}, \tag{18}$$

where T depicts the temperature in the current iteration, and $f(\cdot)$ depicts the objective value function. The temperature T is initialized to the starting temperature at the beginning of the metaheuristic procedure. In each iteration, it is reduced by multiplying it with a cooling rate parameter $\gamma \in (0, 1)$ until eventually a final temperature is reached.

4.5 Local search

Once a new best solution is discovered, a reinsert learner operator is utilized to enhance the solution even more. Initially, this operator identifies all potential relocation moves that can improve the current solution based on the activities already scheduled within the hour being worked on in the iteration. These moves are then executed in descending order of objective gain, with the best moves prioritized. In case a move becomes infeasible due to a previous learner's action, it is skipped. The operator is then applied once more to the improved solution, and this process continues until no additional improvements can be made. Note that this operator also takes into account other hours since it applies the overall utility measure U_{lm} when determining objective gain.

4.6 Updating and operator selection

In every iteration, after an hour has been randomly selected, a roulette wheel mechanism will be applied to determine which of the available destroy and repair operators will be applied for that iteration. This mechanism works as described in (Ropke & Pisinger, 2006). When assuming there are $k \in \mathbb{N}$ operators, with each one having a weight $z_i \geq 0$, $i \in \{1, ..., k\}$, the probability

of selecting operator j is given by

$$\Pr(\text{select } j) = \frac{z_j}{\sum_{i=1}^{k} z_i} \tag{19}$$

We keep track of two sets of weights: one for the destroy operators (ρ_D) and one for the repair operators (ρ_R). Initially, all of the weights are set to 1.

To adjust the weights of the destroy and repair operators, we evaluate their performance based on three possible outcomes: (i) discovering a new best solution, (ii) improving the current solution without affecting the global best solution, and (iii) accepting the solution as new without improving its objective. Each outcome is associated with a factor ω_i, where i ranges from 1 to 3. When a specific operator j produces a certain outcome i, the weights are updated by taking a weighted average of the original weight and the observed outcome. This update is represented by the equation $z_j := \theta z_j + (1 - \theta)\omega_i$. Here, θ is a decay parameter within the range of 0 to 1, controlling the speed at which the metaheuristic reacts to changes in operator effectiveness.

Since it is not possible to differentiate the impact of the destroy and repair operators within a single iteration, both sets of weights are adjusted by the same factor.

For an overview of the code used in this research, see Appendix B.

5 Reproduction of HLAPP results

To analyze the performance of the HLAPP, the same 144 experiments that were run in (Wouda et al., 2023) were used. Firstly, Section 5.1 will be dedicated to the experimental design that was applied to the DLAPP. In Section 5.2, the tuning process will be explained. In Section 5.3, the outcomes of these experiments will be discussed.

5.1 HLAPP experimental design

The parameters used for the analysis can be seen in Table 2.

Table 2: HLAPP experiment parameters and their levels

Parameter	Levels
Self-study penalty parameter (w)	$50\%, 75\%$
Demand spread (σ)	$0, 1, 2, 3$
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 (δ^-)	5
Maximum instruction activity size (δ^+)	30

The experiments provided by Wouda et al.(2023) entail the solving of 100 instances of 144 different experiment types using different parameters. The HLAPP solved by them is identical to the DLAPP when ran for an hour quantity equal to one. When solving the DLAPP for one hour, it is recommended to use the framework of the HLAPP, since it provides less computational strain. The experimental instances provided by Wouda et al.(2023) have defined classrooms, course modules, and teachers which are there to represent a six-year secondary educational

program. This corresponds to regular Dutch education for twelve to eighteen-year-olds. The first parameter which is varied is school size. The schools simulated provide education for 800 up to 1600 learners. The number of teachers is equal to one-tenth of the number of learners, the number of regular classrooms one-twentieth, and the number of large classrooms which can provide self-study for 80 learners come in quantities of three, four, and six respectively. The regular classes are assumed to be able to fit 32 learners. It is always assumed that 12 courses are given in the high school, each one containing 48 modules. The option that classrooms can be split, leading to double the number of rooms that can each only house 16 learners is also explored. The minimum activity size is always equal to 5, the maximum is 30, and various teacher qualification distributions are explored.

The distribution of qualifications among teachers in the set T can be described using a triplet of values $(p, q, 1-p-q)$. Here, p represents the fraction of first-degree teachers, q is the fraction of second-degree teachers, and $1-p-q$ is the fraction of third-degree teachers. These values must satisfy the conditions $p+q \leq 1$, indicating that the combined fractions of first-degree and second-degree teachers cannot exceed the total number of teachers, and $p, q \geq 0$, indicating that the fractions cannot be negative. First-degree teachers can teach all modules, second-degree teachers can only teach the first 24 modules of a course, and third-degree teachers can only teach self-study activities. The penalty which determines the efficiency learners get from learning their most preferred module during a self-study activity is set at either 50% or 75%.

The last parameter which is varied is the demand spread, this parameter indicates whether or not, and at what level, learners can participate in activities in which modules are taught outside of their nominal study schedule. To determine the eligible modules for a learner in a specific year (denoted as y_l, where y_l can be any value between 0 and 5), midpoint calculation is utilized. Assuming the learner follows the standard learning path of completing 8 modules per year, the midpoint μ_l is calculated as $8y_l + 4$. To account for variations in learner eligibilities, σ introduces randomness into the process. This is done by generating a random value from a distribution represented as $\mathcal{N}(\mu_l, \sigma)$, where μ_l is the calculated midpoint and σ represents the degree of randomness. The obtained random value is then rounded to the nearest integer within the range of 1 to 48. This value represents a module a learner can partake in.

All coding was performed in Python 3.9 with Gurobi 10.0.1. The ALNS algorithm was also programmed and solved in Python 3.9 using the ALNS version 4.1.0 package, which is open-source. This is why in this research, the same software was used. The experiments were solved on six Intel i7-9750H 2.6GHz CPU cores with 16GB of ram memory. Unfortunately, memory capacity was significantly smaller than when the problem was originally solved. Since ram capacity was at half of the level of the computer used by Wouda et al.(2023), it was not even possible in this research to solve to optimality for even the smallest school sizes. Due to its purpose of delivering less computational strain on its user, the heuristic fortunately was able to run successfully. However, running all 14.400 cases would still take a very long amount of time, which is why the analysis was conducted using only the first 3 instances of every experiment. This leads to the total number of instances ran being equal to 432. The outcomes of these heuristic solutions were compared to the ILP solutions provided by Wouda et al.(2023). ILP solutions are by definition solved to optimality, which leaves no room for difference in outcome

across runs, except for the instances that were stopped at a time limit that might be imposed.

5.2 Tuning for HLAPP

The tuning of parameters for the metaheuristic was not conducted, since the parameters found by Wouda et al.(2023) could be applied. They computed them as $\omega = (21.8, 13.6, 3.8)$, $\theta = 0.8$, and $d = 15\%$. Furthermore, they applied a starting temperature based on the initial solution, in the same manner as Ropke and Pisinger(2006). Entailing a value such that initially, if a candidate solution were to be five percent worse than the initial solution, it has a chance of 0.5 of being accepted. The cooling rate was set such that it cools to 1 over the 25.000 iterations. Since for the HLAPP experiments performed by Wouda et al.(2023), the heuristic turned out to give better results when excluding the *greedy insert* operator, it was also excluded for the generation of heuristic solutions in the HLAPP experiments of this research. The dataset can be seen in this research file depository as described in Appendix B.

5.3 HLAPP results

Since 4 exact results from the ILP did not have a reported value due to run-time limits imposed by (Wouda et al., 2023)(2023), they and their corresponding heuristic solution were excluded in the analysis, leaving 428 remaining observations.

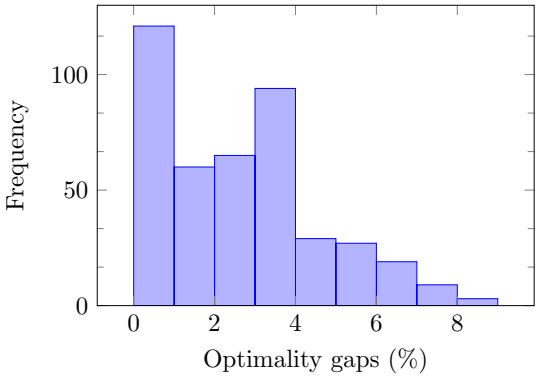


Figure 1: Optimality gap frequencies.

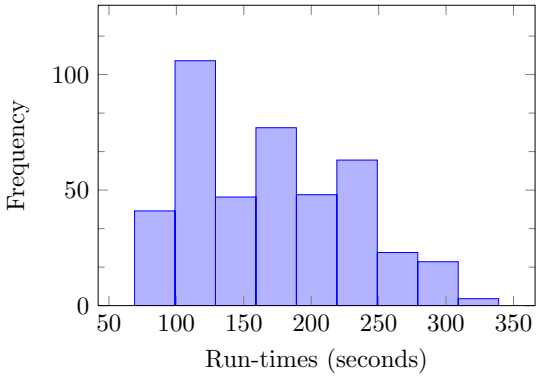


Figure 2: Heuristic run-time frequencies.

When observing the timespans required by the ALNS metaheuristic for solving the HLAPP, the average run-time per instance is about 172 seconds. No instance takes longer than 363 seconds to run, and the fastest one only took 74 seconds. For the exact solutions of the ILP, the average run-time was about 35 minutes. The longest instance took about 12 hours to solve and the fastest run only took 15 seconds. Note that these computation times however are the results of running the algorithm on a different computer than the one used for this research. The performance of the heuristic is evaluated using the optimality gap. This measure is equal to $\frac{f(s^*)-f(s)}{f(s)}$, with $f(s^*)$ being the optimal solution and $f(s)$ the solution of the metaheuristic. Low optimality gaps indicate that the heuristic delivers objective values close to the optimal one, meaning it performs well. The average optimality gap for the 428 instances evaluated is equal to 1.61%. This corresponds to the value of 1.6% reported by Wouda et al.(2023). The frequencies of the optimality gaps and run-times of the 428 heuristic solutions are shown in Figures 1 and

2 respectively.

6 DLAPP experiments

To obtain a dataset that can be used to evaluate the proposed metaheuristic, unfortunately, no database exists to be applied as a benchmark for PL in Dutch secondary education. This is why a data generation process similar to the one used for the HLAPP is applied. This section follows an identical structure as that of Section 5.

6.1 Experimental design

For the experiments conducted to analyze the performance of the ALNS metaheuristic for the DLAPP, a new dataset was generated. The parameters can be observed in Table 3.

Table 3: DLAPP experiment parameters and their levels

Parameter	Levels
Self-study penalty parameter (w)	50%
Number of hours ($\|H\|$)	$2, 3$
Monotonicity penalty (Λ)	$0\%, 5\%, 20\%$
Demand spread (σ)	$0, 0.2$
School size (# learners)	200
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.
Minimum activity size (δ^-)	1
Maximum instruction activity size (δ^+)	30

As can be seen in the table, two new parameters that were added and varied to analyze heuristic performance are $|H|$ and Λ. Note that all of the possible combinations between Λ and $|H|$ adhere to the tight restriction from Lemma 2. The total number of possible combinations between the parameters is equal to 36. Note that in contrast to the experiments of the HLAPP, school size was not varied since computational limitations require it to remain small. Learners and modules have quantities in the same proportions to learners as in the HLAPP, and only one large self-study classroom which can house 80 students has been created. The option of splitting classrooms was also not explored in this research, since it increases experiment quantities, and is not a well-implemented option in Dutch secondary education. Due to the very similar nature of the DLAPP ALNS metaheuristic to the algorithm for the HLAPP, it is unlikely that a split in classrooms leads to a significant change in the optimality gap or run-time, which is the main focus of the initial analysis of the DLAPP. Differing self-study penalties were also not explored due to computational limitations. For future research though, it may prove interesting to analyze the exact scheduling performance differences and policy implementations when varying school sizes, self-study penalties, or when allowing classrooms to be split.

As shown in Wouda et al.(2023), higher demand spread values σ cause the ILP to take much longer to solve. This effect is even stronger when solving the DLAPP since the higher complexity of the problem causes stronger exponential growth of computation time in parameters such as school size or σ. This causes the solving of the ILP formulated in Section 3 to take well over ten hours for an instance with a σ equal to 1 and an hour quantity of 3 during the experimentation

phase. This very high run-time that does not allow for analysis over many instances due to time limitations occurs even though the school size has already been drastically reduced to 200 learners to make the ILP faster to solve. However, a non-zero σ value is very interesting to analyze since it allows learners to participate in modules outside of their nominal study schedule. This is why a very small non-zero value of σ, namely 0.2, was also experimented with.

During the experimentation phase, a problem concerning the minimum activity size arose. A last resort of the greedy insert operator when no bigger classroom can be chosen for an existing activity, no activity can be split, and there is no self-study activity available, is to create a new self-study activity by taking a number of unassigned learners equal to the minimum activity size and inserting them into this new activity. This however causes an error when there are not sufficient learners to create this activity while abiding by the minimum activity size. This problem never occurred for the HLAPP, however, it is a problem when applying the earlier-mentioned minimum activity size of 5 learners to the DLAPP. This was likely caused by the reduced school size. If the number of learners is smaller, there are also fewer learners being unassigned in every iteration of the heuristic. This decreases the chance that there are enough unassigned learners to create a new self-study activity when needed whilst abiding by the minimum activity size.

Because of this issue, the minimum activity size has been decreased to one for the experiments in this research. Applying the exact parameters mentioned in Table 2 however might prevent this issue even for the DLAPP, meaning that applying $\delta^- > 1$ might be possible without errors for this case. For all of the 36 possible experiments that could be made using these parameters, 3 instances were created, leading to a total of 108 instances ran. Not that all teacher and classroom qualifications were generated in the same manner as by Wouda et al.(2023). An overview of the experiments and their parameter settings can be seen in Table 5 (Appendix C).

6.2 Tuning

The tunable parameters were taken at an identical level as in Section 5.2 because of the similar nature of the HLAPP and DLAPP. Where Wouda et al.(2023) managed to run their HLAPP ALNS procedure using 25000 iterations, with all instances being solved within ten minutes, the DLAPP proves to deliver significantly higher computation times. The cause of this is probable to be the fact that the DLAPP constantly has to update the current, the positive hypothetical, and the negative hypothetical utility a learner gets from a model by taking it in the current, current $+1$, or current -1 quantity. This is required to be able to quickly compute possible changes in utility levels. Because of this increased computational strain and the desire to be able to rapidly obtain a solution, the number of iterations has been reduced to 500. However, since the exclusion of some operators appeared to lead to average optimality gaps not that far from each other, and since these operators have changed, analysis is conducted to determine whether the exclusion of an operator leads to better performance.

To analyze this, of the 3 instances generated for each of the 36 experiments only the first one was initially used. These 36 single instances were solved to optimality using the ILP, and then they were solved for 8 possible configurations of the metaheuristic. One incorporates the full set of operators from Section 4, then there are 6 configurations which each exclude one of

the destroy or repair operators. The last one excludes the *local search* algorithm.

Some statistics on the optimality gaps for the 8 configurations are shown in Figure 3 and Table 4.

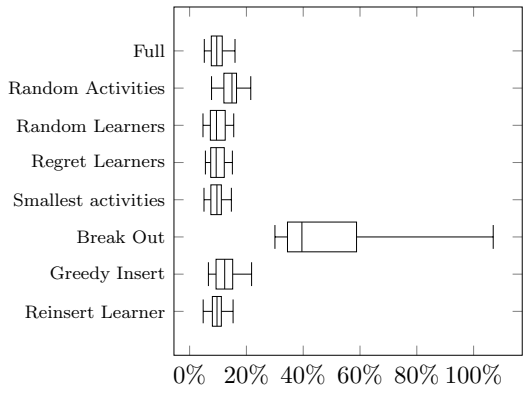


Figure 3: Box-and-whisker plots of optimality gaps when excluding certain operators.

Table 4: Average gaps and run-times when excluding certain operators.

Operator	**Average Gap (%)**	**Average run-time (Seconds)**
Full	9.80	134.6
Random activities	14.52	113.96
Random learners	9.69	142.1
Regret learners	9.56	141.7
Smallest activities	9.61	142.69
Break out	50.91	43.9
Greedy insert	12.58	168.7
Reinsert learner	9.51	140.82

As can be seen in the figure and the table, excluding a repair operator causes a clear change in heuristic performance. Removing either *greedy insert* or *break out* causes a notable increase in the average optimality gap. Especially the *break out* operator turns out to be vital for the heuristic. For the *reinsert learner* algorithm and any of the destroy operators, not a large change in optimality gap can be observed.

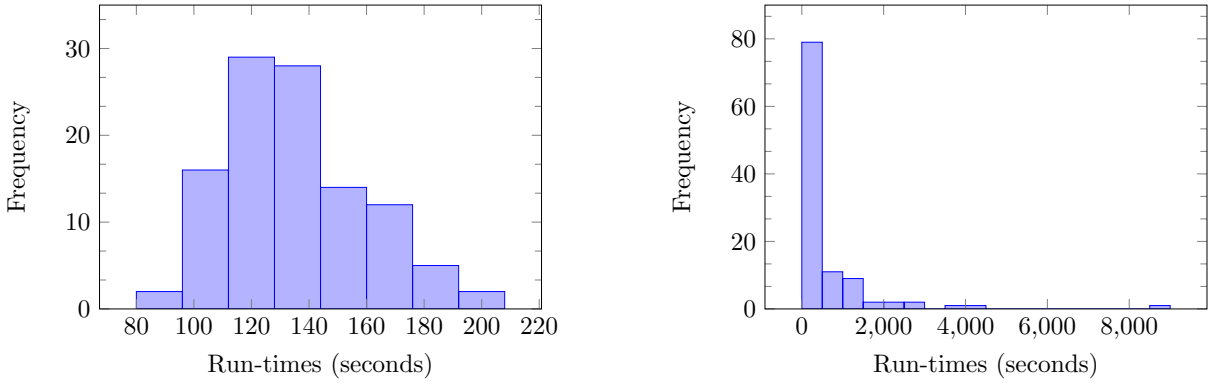
Since the sample size of 36 instances is quite low, the other differences in the average gap do not seem to be significant enough to exclude any operator for the rest of the research. The best operator that could be removed would be *reinsert learner*, which has an average optimality gap that is 0.29 percentage points smaller than the one of the full set of operators.

The gap does not appear to be reduced by a significant proportion when excluding any operators. This is supported by a simple t-test with the null hypothesis that the average difference in optimality gaps when either not excluding, or excluding *reinsert learner* is 0, and the alternative hypothesis is that the difference is larger than 0. This delivers a p-value of 0.23, meaning we do not reject the null hypothesis for any significance value at or below 10%. Based on this finding and the desire to explore the performance of the heuristic with all mutated operators included, the rest of the results for this initial analysis were generated with the full set of operators applied. Tables 7 and 8 (Appendix D) provide an overview of all heuristic outputs for the operator exclusion options. Additionally, Table 6 in the same appendix displays the differences in gaps used for the t-test.

6.3 Metaheuristic performance

The metaheuristic is compared to the exact solution of the ILP proposed in Section 3 using the experiments generated as explained in Section 6.1. The heuristic can solve all of the instances quickly, with a maximum run-time of 198 seconds. The average run-time across all instances is 135 seconds, and the minimum run-time is equal to 81 seconds. An overview of the run-times of the heuristic can be seen in Figure 4a.

The run-times of the ILP vary widely. The fastest experiments, which consist of the smallest number of hours $|H| = 2$ and a Λ level of 0 are solved very rapidly, in around 15 seconds. This is considerably faster than the heuristic. However, as $|H|$ and Λ increase, the solving time grows explosively. The instance that took the longest to run was an experiment consisting of 3 hours, a σ of 0.2, and a Λ level of 20%. This instance took 8805 seconds. The solving time of the ILP will explosively grow further as the number of hours or the value of σ increases even further, indicating the high value of a well-performing heuristic. An overview of the run-times for the exact solutions is given in Figure 4b. In this graph, it can be observed that a very large part of observations is in the smallest run-time category, which ranges from 0 to 500 seconds. Only a small part of observations fall into the high-run-time categories. However, note that the seven run-times above 2000 seconds make up for about 48.6% of the sum of all run-times for the ILP.

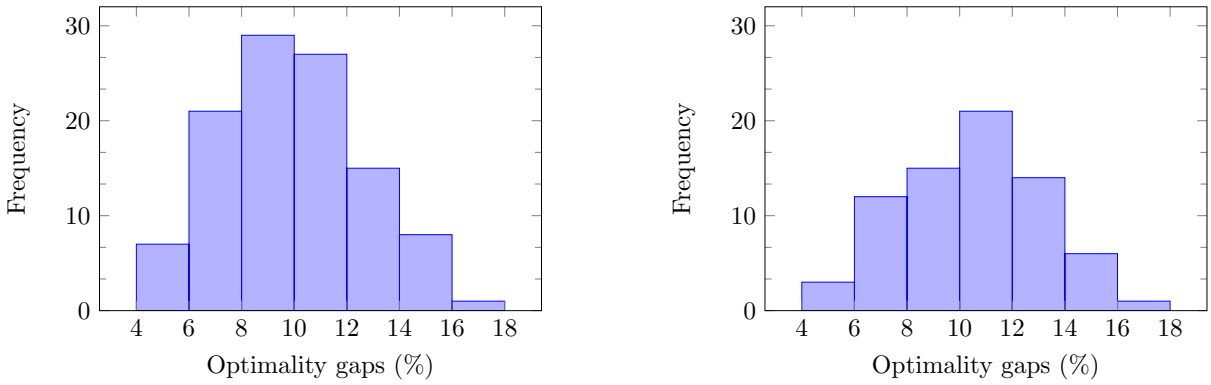


(a) Run-time frequencies of heuristic.

(b) Run-time frequencies of ILP.

Figure 4: Comparison of run-time frequency distributions for DLAPP instances.

For the DLAPP, the optimality gaps are computed identically as for the HLAPP, using a gap equal to $\frac{f(s^*)-f(s)}{f(s)}$, with $f(s^*)$ being the optimal solution and $f(s)$ the solution of the adjusted metaheuristic. The average gap in the results is about 9.88%. The maximum is 16.22% and the lowest observed gap value is 4.90%. The distribution of the optimality gaps can be seen in Figure 5a. An overview of the objectives, run-times, and optimality gaps of all 108 instances can be found in Tables 9 and 10 in Appendix E.



(a) All 108 instances.

(b) Only instances with $\Lambda > 0$.

Figure 5: Optimality gap frequencies of DLAPP using 500 iterations.

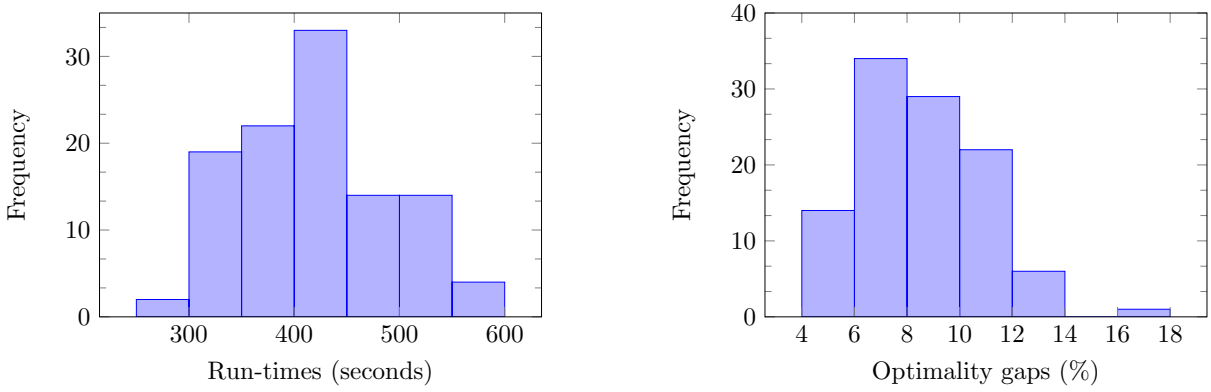
Since the $\Lambda = 0$ case can be more efficiently solved by the HLAPP ALNS metaheuristic

because of the aforementioned reasons, it is only fair to also exclusively analyze the performance in cases where applying the renewed metaheuristic is essential, namely when $\Lambda > 0$. When only looking at the 72 experiments in which this is the case, the mean optimality gap is 0.5 percentage points larger, having an average value of 10.39%. The minimum value of the gaps in these instances is 5.54%, and the highest gap instance appears to be in this set and remains at 16.22%. This highest gap instance occurs in experiment 3, which has a Λ value of 20% and an hour quantity of 2. The other highest gap instances also appear to fall into this category.

To explore the improvement of heuristic performance when iteration quantities are raised, all 36 experiments were also run at an iteration quantity of 2000. Running the ALNS algorithm at this setting leads to an improvement in the average optimality gap. The optimality gap shrunk by 1.38 percentage points to 8.50%. The largest gap, which is equal to 16.10%, occurs in instance 9, which similarly to experiment 3 has a Λ of 20% and a $|H|$ of 2. The smallest gap is equal to 3.88%. This smallest gap occurs in experiment 29, which has a Λ of 5% and an hour quantity of 3.

When excluding the zero-monotonicity penalty case, the increase in iterations led to an average optimality gap of 9.20%. The minimum and maximum gaps are unchanged since they both come from $\Lambda > 0$ experiments. An overview of run-times and optimality gap frequencies for the 2000 iteration solutions are shown in Figures 6a and 6b respectively.

The full outputs concerning objective values, run-times, and optimality gaps of all these 108 instances can be found in Tables 9 and 10 in Appendix E.



(a) Run-time frequencies.

(b) Optimality gap frequencies.

Figure 6: 2000 iteration heuristic results (all 108 instances).

For all experiments that were tested, the increased iteration quantity showed a promising decrease in average optimality gap, though average run-time was increased to 418 seconds. This illustrates the potential the metaheuristic has if it were allowed to execute even more iterations. This, however, should be executed by stronger hardware, since short-term timetabling cannot allow for run-times to become significantly longer than ten minutes.

Even after increasing the number of iterations to 2000, pushing the limits of what is time-technically feasible, the heuristic performance still appears to be significantly worse than that of the ALNS metaheuristic for the HLAPP, which only had a gap of 1.61%.

The first reason why this might be the case is that the number of iterations had to be drastically reduced due to higher computational strain. The number of iterations applied, after

increasing it to 2000, is still only 8% of the number of iterations used for the HLAPP. One could argue that this fraction is even smaller in practice. The reason for this is that one iteration only adjusts the solution for one hour. This means that the number of iterations per hour is only equal to $\frac{|H|}{2000}$, while the ALNS for the HLAPP can devote all of its 25.000 iterations to the single hour it has to work on. A second reason for the DLAPP heuristic performing worse than the one for the HLAPP is the more complex nature of the problem. In one iteration, an hour is taken, and within that hour, the heuristic looks at the potential moves a learner can make, whilst taking the current, positive hypothetical, and negative hypothetical utility into account utilizing Equation 16. This causes the (group of) learners to, for that iteration, be reallocated as well as possible given the current solution state. However, this 'greedy' reallocation process does not take into account any possible future moves within the solution. A new solution that maximizes the increase in overall utility for the hour being worked on in the current iteration might block a potential future allocation that intelligently plans activities, with the activities compromising their own utility so that it can be gained in places with more potential. The procedure does have features to lower the chance of getting stuck in local maxima, but the model might still be overwhelmed by the increased complexity of the problem.

7 Conclusion

In conclusion, this research introduces the daily learning activity planning problem (DLAPP) and proposes a tailored adaptive large neighborhood search (ALNS) metaheuristic as a practical and efficient approach to solving this scheduling problem in educational institutions. It extends the hourly learning activity planning problem (HLAPP) proposed by Wouda et al.(2023) for personalized learning in Dutch education. The novel scheduling problem involves assigning learners to activities, teachers, classrooms, and modules for every hour in a schoolday, taking into account learner demands and reduced utility when facing a monotonous schoolday.

One of the key challenges in the scheduling problem is the increasing complexity of interdependence as the number of hours to be scheduled increases. This complexity makes it extremely computationally demanding for schools to optimally plan for more than a few hours. The ALNS algorithm addresses this challenge by providing a systematic and effective method to quickly renew the schedule for the remaining hours, enabling schools to generate feasible schedules within a reasonable time span.

The ALNS algorithm begins by constructing a simple, feasible initial solution. It then proceeds through iterations, randomly selecting an hour to work with and applying destroy and repair operators to transform the current solution into a new candidate solution. The candidate solution is evaluated, and if it is accepted, a local search mechanism is employed to further refine the solution. This iterative process continues until a maximum number of iterations is reached, ultimately returning the best-found solution.

This paper delves into several important aspects of the ALNS algorithm. It discusses the construction of the initial solution, which forms the foundation for subsequent improvements. It explores various destroy operators that selectively remove learners from the solution, as well as repair operators that assign unassigned learners to appropriate modules. This paper also covers the acceptance criterion based on simulated annealing, which allows the algorithm to explore

potentially suboptimal solutions to avoid getting trapped in local maxima. Additionally, the updating and operator selection mechanism ensure adaptability and fine-tuning of the algorithm based on the specific characteristics of the problem.

The experimental evaluations conducted show promising results, indicating the effectiveness of the ALNS metaheuristic in generating decent-quality schedules to solve the DLAPP. The optimality gap the metaheuristic provides is 10.39% on average when compared to the computed optimal solutions in cases where the HLAPP cannot be applied. Pushing the limits of computation time can even get the optimality gap as low as 8.50% when evaluating the entire set of experiments. Experimentation has shown that optimality gaps have the potential to become even lower when excluding certain operators from the ALNS algorithm.

Overall, the DLAPP forms a problem definition worthy of further research. The corresponding ALNS metaheuristic presents a promising solution for educational institutions to efficiently handle complex scheduling problems. By offering flexibility, adaptability, and the ability to generate schedules for multiple hours, the ALNS algorithm empowers schools to meet their specific scheduling needs effectively. As technology advances and computational resources become more powerful, the DLAPP and the metaheuristic tailored to solving it have the potential to become indispensable in optimizing educational scheduling processes in a world where personalized learning becomes more and more important.

8 Limitations and future research

Because of the large potential the DLAPP has in learner-centered scheduling, ample space is left for future research. Firstly, computational capacity was strongly limited in this research. The HLAPP which was extended could not even be optimally solved for schools with eight hundred learners due to ram-memory limitations. This means the DLAPP has only been solved and evaluated for very small schools with only 200 learners so far. Larger hour quantities were also not analyzed for this reason. Lastly, more spread-out eligibilities for modules could not be explored, since a larger spread rapidly increases computational time towards the optimal solution. Stronger hardware will also be able to run the ALNS metaheuristic for more iterations within the few minutes it is allowed to take.

Apart from the computational limitations, some technical ones are also present in this paper. Firstly, no new hyperparameter tuning procedure was applied, and these values instead were instead taken from the results of the HLAPP computed by Wouda et al.(2023). Expertly tuning these parameters might prove for providing an improvement in metaheuristic results for the DLAPP. Furthermore, though the ALNS procedure was properly adapted, the high complexity of multiple-hour planning proves to limit when greedy-improvement algorithms are applied. More complex algorithms might prove even more proficient in preventing embeddedness in local maxima that prevent the localization of more potent ones.

Lastly, future research could expand the problem definition and adjust the solution approach to further lower the distance the model has to real life. Self-study penalties could be varied per learner, and the feature that learners could choose to study different subjects in different self-study activities to decrease monotonicity could also be explored. Further analysis could also form a basis for policy advice that can be provided to schools.

Acknowledgements

I would like to thank Danny Zhu for his guidance and advice during the writing of this paper. I would also like to thank him and Prof. Dr. Dennis Huisman for the evaluation of this research.

References

Brown, B. W. & Saks, D. H. (1987). The microeconomics of the allocation of teachers' time and student learning. *Economics of education review*, *6*(4), 319–332.

Commission, E. & Centre, E. P. S. (2019). *10 trends transforming education as we know it.* Publications Office. doi: doi/10.2872/800510

Eiken, O. (2011). The kunskapsskolan ("the knowledge school"): a personalised approach to education.

Faryadi, Q. (2017). The application of montessori method in learning mathematics: An experimental research. *Open Access Library Journal*. doi: 10.4236/OALIB.1104140

Foundation, T. M. (2022). *The international montessori council.* Retrieved from `https://www.montessori.org/`

Hiles, E. (2018). Parents' reasons for sending their child to montessori schools. doi: 10.17161/JOMR.V4I1.6714

Kunskapsskolan. (2023, Mar). *Wie is kunskapsskolan nederland.* Retrieved from `https://kunskapsskolan.nl/wie-zijn-wij/`

Marshall, C. (2017). Montessori education: A review of the evidence base. *npj Science of Learning*, *2*(1), 11.

McCarthy, E. M., Liu, Y. & Schauer, K. L. (2020). Strengths-based blended personalized learning: An impact study using virtual comparison group. *Journal of Research on Technology in Education*, *52*(3), 353–370.

Ropke, S. & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, *40*(4), 455–472.

Santini, A., Ropke, S. & Hvattum, L. M. (2018). A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. *Journal of Heuristics*, *24*, 783–815.

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.

Appendices

A Proofs

A.1 Proof of Lemma 1

Lemma 1. *Let Λ be the monotonicity penalty measure applied to the utility learner l gains from module m in a day with $|H| > 1$ hours in it. If a learner l should gain positive utility from module m by taking it any number of times that day, $\Lambda \leq \frac{1}{|H|-1}$.*

Proof. Consider a schedule with $|H| > 1$ in which a learner l and module m exist with a level of productivity

$$p_{lm} = 1 + (1 - \sum_{h \in H} y_{lmh})\Lambda.$$

If all hours are spent on module m by learner l,

$$\sum_{h \in H} y_{lmh} = |H| \implies p_{lm1} = 1 + (1 - |H|)\Lambda.$$

Since $D_{lm} \geq 0$ and $y_{lmh} = 1$, the utility learner l gains from module m

$$u_{lm} = D_{lm} p_{lm1} \sum_{h \in H} y_{lmh}$$

would become negative if $p_{lm1} < 0$, therefore,

$$p_{lm1} \geq 0 \implies \Lambda \leq \frac{1}{|H| - 1}.$$

Any case where $|H|$ would be replaced with a lower quantity of hours would lead to a less tight restriction on Λ, ending the proof. □

A.2 Proof of Lemma 2

Lemma 2. *Let Λ be the monotonicity penalty measure applied to the utility learner l gains from module m in a day with $|H| > 1$ hours in it. If a learner l should always gain more utility by partaking in module m one more time that day, $\Lambda \leq \frac{1}{2|H|-1}$.*

Proof. Consider a schedule with $|H| > 1$ in which a learner l and module m exist with a level of productivity

$$p_{lm} = 1 + (1 - \sum_{h \in H} y_{lmh})\Lambda$$

and level of utility learner l gains from solely module m

$$u_{lm} = D_{lm} p_{lm} \sum_{h \in H} y_{lmh}.$$

If all hours are spent on module m by learner l,

$$\sum_{h \in H} y_{lmh} = |H| \implies p_{lm1} = 1 + (1 - |H|)\Lambda.$$

If the learner were to spend one less hour on the module,

$$\sum_{h \in H} y_{lmh} = |H| - 1 \implies p_{lm2} = 1 + (2 - |H|)\Lambda.$$

Then, if total utility should be higher when the number of times the module is taken increases,

$$D_{lm}p_{lm1}|H| \geq D_{lm}p_{lm2}(|H|-1) \implies \Lambda \leq \frac{1}{2|H|-1}.$$

Any case where $|H|$ would be replaced with a lower quantity of hours would lead to a less tight restriction on Λ, ending the proof. □

Note that in these proofs, u_{lm} is not the same as U_{lm} from Equation 16. The latter represents the sum of utility gained from all modules, whereas the prior only indicates the utility gained from the particular module m. This means that $\sum_{m \in M} u_{lm} = U_{lm}$

B Code description

The code used to compute the outcomes of the DLAPP consists of an extension of the Python 3.9 code provided by Wouda et al.(2023). The HLAPP code was not adjusted in any way that changes functionality. The HLAPP, as well as the DLAPP map that were added contain a set of experiments with generated data, outputs, a cache containing analyzed results, and a src (source) map with the code that was used.

The original source map of the HLAPP that was extended on contains the following maps and scripts:

- Classes. This map contains objects which can be used
  - Activity.py. One activity in the solution and its properties.
  - Classroom.py. A classroom from the problem and its properties.
  - Learner.py. A learner with an ID and a year.
  - Module.py. One of the 577 modules that exist.
  - Problem.py. Provides the problem, some cached properties for quick access, and some operators.
  - Result.py. Result object that can be formed from a (potential) solution.
  - Solution.py. One possible solution. Contains activities, currently unassigned learners, used classrooms, and used teachers.
  - Teacher.py. A teacher and their degree.
- Destroy operators. This map contains the set op destroy operators. Operators work as described by Wouda et al.(2023).
  - random activities.py.
  - random learners.py.
  - regret learners.py.
  - smallest activities.py.
- Functions. A map containing some baseline functions of the ALNS or ILP algorithm.

  - initial solution.py. Creates initial solution with only self-study activities
  - learners to remove.py. Determines the number of learners that will be removed based on the degree of destruction
  - pairwise.py. A function that helps symmetry-breaking constraints for the ILP.
  - problem.py. Gets or sets the problem

- Local Search. This map contains the reinsert learner algorithm.
  - reinsert learner.py. Works as described by Wouda et al.(2023).

- Repair operators. A map containing the two repair operators. Operators work as described by Wouda et al.(2023).
  - break out.py.
  - greedy insert.py.

- Rules. A map containing all of the functions from the model in Section 3.
- analyse.py. This script analyses the results of an experiment run.
- constants.py. This script defines hyperparameters for the ALNS heuristic.
- heuristic.py. Runs heuristic algorithm. Then prints and stores its results.
- ilp.py. Runs ILP algorithm. Then prints and stores its results.
- make experiments.py. Makes multiple instances for all experiments based on possible parameter settings. (Not used for reproduction).
- tune.py. Tunes hyperparameters. (Not used or adjusted at any point in this research).
- validator.py. Validates outputs.

To make the computation of results more convenient, some 'heuristic loop.py', 'ilp loop.py', and 'analyse loop.py' classes have been added to the HLAPP and DLAPP map to iteratively run the respective scripts, with the experiment, instance, and potentially excluded operator given as argument. The loops explain in comments what they can be used for. a 'constants2.py' script has also been made to be able to run the 2000 iteration heuristic through 'heuristic loop3.py' without needing to manually adjust the quantity.'heuristic loop 2' performs the runs concerning the exclusion of operators. Separate maps have been created for the outputs and analysis of 2000 iterations and exclude operator runs.

Since the outputs made by the analyze class for analysis are stored in different Excel files for every experiment, a 'mergeCache.py' script was made to put them all into one Excel file. These 'all results' Excel files were used for data analysis by the author, and need not be observed to extract the findings of this paper.

For the code used to obtain results for the DLAPP, the code of the HLAPP environment has been adjusted in many ways. The most important changes have been described below. Firstly, for the exact solution of ILP, the 'ilp.py' script has been adjusted to implement the model from

Section 3. Secondly, many parts of the code for the heuristic have been adjusted to implement the solution approach from Section 4.

The only script which has been added for the heuristic is the Hour.py script within the classes map which represents an hour object. For coding purposes, hours are numbered from 0 to $|H| - 1$.

The problem class now also stores all possible utilities for all learner module combinations for every hour quantity possible, taking the monotonicity penalty Λ into account.

The solution class now also stores the number of times each learner takes each module at all times, as well as the corresponding utility a learner currently gains from a module. The key to the application of the ALNS framework on the DLAPP is that instead of using general preferences D_{lm} and differences in them to compute what moves can best be made, learner utilities U_{lm} should be used instead. At all times, when it needs to be computed what would happen to the utility if a learner were to take a module one more or one fewer times in the schedule, the hypothetical utility a learner would have in that case needs to be accessed. Instead of newly computing this whenever needed using the current assignments (which causes large computational strain), the solution object not only always keeps track of the current utility each learner gains from each module, but also the hypothetical positive utility and hypothetical negative utility. These values can quickly be accessed whenever needed by simply requesting them using the learner id and module id. The learner-module quantities, the current utility, and the two hypothetical utilities are changed any time a change in assignment occurs (if it affects learner-module quantities).

The four destroy operators, the repair operator and the reinsert learner algorithms have been adjusted so that they select a random hour to be worked on in every single iteration, and can then only change the solution by working within that hour. The random learners, random activities, and smallest activities operators have not been adjusted apart from this, since just like in the HLAPP, they should unassign learners without taking the objective into account.

The regret learners, break-out, greedy-insert and reinsert learner algorithms however all take into account the objective function when determining their mutations. This is why every single case in which preferences were used was adequately adjusted to work with current and hypothetical utilities instead. Every single method within each object class that is used when the objective is taken into account has been adjusted to properly incorporate current and hypothetical utilities. These changes are described in comments in these classes.

A short script called 't test' has also been added to perform the t-test from section 6.2.

Lastly, the 'validator.py' has been adjusted to analyze the feasibility of a DLAPP solution. The 'validator loop' script loops this validator over different experiments and instances to evaluate all of them in one go. All restrictions from rules have been properly adjusted, and the rule to check if the penalty measure is well-enforced has been added. executing the validator loop has shown that all generated solutions are feasible.

C Experiment parameters

Table 5: Parameter settings of all experiments in which instances have been generated

Exp.	#Learners	w	σ	Λ	$\lvert H\rvert$	Split?	$(p, q, 1-p-q)$
1	200	50%	0	0%	2	No	(1, 0, 0)
2	200	50%	0	5%	2	No	(1, 0, 0)
3	200	50%	0	20%	2	No	(1, 0, 0)
4	200	50%	0	0%	3	No	(1, 0, 0)
5	200	50%	0	5%	3	No	(1, 0, 0)
6	200	50%	0	20%	3	No	(1, 0, 0)
7	200	50%	0	0%	2	No	(0.5, 0.5, 0)
8	200	50%	0	5%	2	No	(0.5, 0.5, 0)
9	200	50%	0	20%	2	No	(0.5, 0.5, 0)
10	200	50%	0	0%	3	No	(0.5, 0.5, 0)
11	200	50%	0	5%	3	No	(0.5, 0.5, 0)
12	200	50%	0	20%	3	No	(0.5, 0.5, 0)
13	200	50%	0	0%	2	No	(0.4, 0.4, 0.2)
14	200	50%	0	5%	2	No	(0.4, 0.4, 0.2)
15	200	50%	0	20%	2	No	(0.4, 0.4, 0.2)
16	200	50%	0	0%	3	No	(0.4, 0.4, 0.2)
17	200	50%	0	5%	3	No	(0.4, 0.4, 0.2)
18	200	50%	0	20%	3	No	(0.4, 0.4, 0.2)
19	200	50%	0.2	0%	2	No	(1, 0, 0)
20	200	50%	0.2	5%	2	No	(1, 0, 0)
21	200	50%	0.2	20%	2	No	(1, 0, 0)
22	200	50%	0.2	0%	3	No	(1, 0, 0)
23	200	50%	0.2	5%	3	No	(1, 0, 0)
24	200	50%	0.2	20%	3	No	(1, 0, 0)
25	200	50%	0.2	0%	2	No	(0.5, 0.5, 0)
26	200	50%	0.2	5%	2	No	(0.5, 0.5, 0)
27	200	50%	0.2	20%	2	No	(0.5, 0.5, 0)
28	200	50%	0.2	0%	3	No	(0.5, 0.5, 0)
29	200	50%	0.2	5%	3	No	(0.5, 0.5, 0)
30	200	50%	0.2	20%	3	No	(0.5, 0.5, 0)
31	200	50%	0.2	0%	2	No	(0.4, 0.4, 0.2)
32	200	50%	0.2	5%	2	No	(0.4, 0.4, 0.2)
33	200	50%	0.2	20%	2	No	(0.4, 0.4, 0.2)
34	200	50%	0.2	0%	3	No	(0.4, 0.4, 0.2)
35	200	50%	0.2	5%	3	No	(0.4, 0.4, 0.2)
36	200	50%	0.2	20%	3	No	(0.4, 0.4, 0.2)

D Exclude operators outputs

Table 6: Difference in optimality gaps from 36 single-instance experiments, calculated as: (optimality gap when excluding *reinsert learner*) - (optimality gap with all operators included).

Experiment	1	2	3	4	5	6	7	8	9	10	11	12
Diff in Opt. gap	-2.73	1.09	4.82	-3.15	1.1	-2.82	3.12	1	-2.04	-0.27	1.3	0.41
Experiment	13	14	15	16	17	18	19	20	21	22	23	24
Diff in Opt. gap	-2.78	0.57	0.66	0.39	-0.22	-2.04	-0.06	-0.92	1.77	-1.11	0.97	2.72
Experiment	25	26	27	28	29	30	31	32	33	34	35	36
Diff in Opt. gap	0.19	-1.66	1.96	0.07	-1.21	0.25	7.87	-0.57	2.36	2.73	-0.52	-2.86

Table 7: Objectives, run-times and optimality gaps of ALNS metaheuristic when excluding certain operators, or when none are excluded (part 1).

			Exclude	**reinsert**	**learner**	**Exclude**	**none**		**Exclude**	**greedy**	**insert**	**Exclude**	**break**	**out**
Exp.	**Inst.**	**ILP Objective**	**ALNS Objective**	**Run-time**	**Opt. Gap**	**ALNS Objective**	**Run-time**	**Opt. Gap**	**ALNS Objective**	**Run-time**	**Opt. Gap**	**ALNS Objective**	**Run-time**	**Opt. Gap**
1	1	1619.05	1453.06	113.53	11.42	1489.61	113.41	8.69	1470.78	185.01	10.08	1218.07	38.66	32.92
2	1	1561.15	1420.68	134.57	9.89	1406.72	110.2	10.98	1387.87	163.62	12.48	1160.9	40	34.48
3	1	1621.2	1473.23	163.24	10.04	1411.4	133.12	14.86	1398.07	199.25	15.96	1020.65	39.65	58.84
4	1	2589.87	2390.89	146.28	8.32	2462.53	139.87	5.17	2406.2	170.22	7.63	1985.25	41.71	30.46
5	1	2383.8	2263.29	184.45	5.32	2240.03	157.96	6.42	2151.73	202.46	10.79	1715.17	49.52	38.98
6	1	2314.8	2063.06	170.39	12.2	2116.26	168.7	9.38	2008.3	196.03	15.26	1132.64	46.31	104.37
7	1	1675.02	1548.36	150.78	8.18	1504.97	121.47	11.3	1497.25	155.82	11.87	1258.24	48.03	33.12
8	1	1483.65	1322.9	117.41	12.15	1311.27	97.31	13.15	1296.12	155.56	14.47	1089.63	47.26	36.16
9	1	1648.79	1429.93	133.2	15.31	1455.67	127.33	13.27	1353.44	131.15	21.82	1029.81	44.48	60.11
10	1	2494.7	2320.12	151.97	7.52	2326.16	125.61	7.25	2301.22	153	8.41	1891.16	46.96	31.91
11	1	2353.99	2170.96	166.3	8.43	2145.32	122.35	9.73	2150.06	161.45	9.49	1653.02	40.83	42.41
12	1	2287.4	2082.62	141.91	9.83	2074.94	161.7	10.24	1944.58	149.7	17.63	1143.17	42.33	100.09
13	1	1549.63	1402.93	120.07	10.46	1439.05	109.89	7.68	1399.79	145.46	10.7	1176.73	41.91	31.69
14	1	1610.92	1469.96	148.75	9.59	1462.32	128.74	10.16	1403.53	142.3	14.78	1184.67	38.36	35.98
15	1	1528.37	1357.98	181.08	12.55	1350.09	136.51	13.21	1268.98	139.14	20.44	962.69	40.23	58.76
16	1	2450.79	2233.93	184.1	9.71	2225.96	112.23	10.1	2298.65	170.97	6.62	1823.25	42.15	34.42
17	1	2336.91	2187.66	168.64	6.82	2192.14	133.33	6.6	2178.62	182.62	7.27	1654.7	51.93	41.23
18	1	2176.74	1930.04	134.94	12.78	1965.6	129.79	10.74	1878.89	156.41	15.85	1052.05	46.82	106.91
19	1	1563.45	1435.03	113.13	8.95	1435.81	98.95	8.89	1393.23	154.11	12.22	1202.24	45.53	30.04
20	1	1601.72	1474.14	110.95	8.65	1486.79	118.21	7.73	1419.88	137.9	12.81	1171.56	49.95	36.72
21	1	1528.14	1375.48	145.25	11.1	1353.91	122.67	12.87	1332.52	146.38	14.68	978.8	44.62	56.12
22	1	2603.86	2416.4	132.23	7.76	2441.52	119.73	6.65	2398.94	168.65	8.54	1905.05	50.97	36.68
23	1	2359.98	2202.69	141.43	7.14	2182.84	130.65	8.11	2169.48	179.81	8.78	1719.97	47.09	37.21
24	1	2243.21	2013.95	156.87	11.38	1966.07	134.12	14.1	1948.29	153.24	15.14	1110.31	46.07	102.03
25	1	1640.36	1543.02	128.82	6.31	1540.18	104.78	6.5	1493.69	134.87	9.82	1218.34	47.55	34.64
26	1	1591.13	1442.42	99.55	10.31	1464.45	118.3	8.65	1399.86	144.98	13.66	1135.4	45.84	40.14
27	1	1564.75	1420.9	170.83	10.12	1396.08	142.37	12.08	1326.27	172.75	17.98	1014.49	46.17	54.24
28	1	2411.16	2230.46	128.1	8.1	2229.07	145.87	8.17	2220.26	202.93	8.6	1825.11	47.19	32.11
29	1	2418.66	2265.78	155.83	6.75	2291.8	188.49	5.54	2267.21	222.78	6.68	1701.65	47.7	42.14
30	1	2177.08	1960.38	123.97	11.05	1956.11	164.3	11.3	1932.97	250.83	12.63	1066.23	44.69	104.19
31	1	1686.96	1560.25	112.45	8.12	1454.44	140.01	15.99	1443.06	159.65	16.9	1262.41	38.53	33.63
32	1	1625.9	1490.37	117.7	9.09	1498.28	181.98	8.52	1465.19	191.33	10.97	1160.55	37.35	40.1
33	1	1559.24	1386.46	100.22	12.46	1358.01	127.37	14.82	1296.9	170.44	20.23	979.92	39.06	59.12
34	1	2525.84	2410.7	153.09	4.78	2349.34	142.95	7.51	2348.52	175.86	7.55	1876.31	38.53	34.62
35	1	2381.52	2240.81	144.09	6.28	2251.78	178.03	5.76	2160.96	177.26	10.21	1651.63	38.51	44.19
36	1	2250.51	1981.36	123.6	13.58	2032.59	156.45	10.72	1974.72	167.91	13.97	1113.23	39.34	102.16

Table 8: Objectives, run-times and optimality gaps of ALNS metaheuristic when excluding certain operators, or when none are excluded (part 2).

			Exclude	**random**	**activities**	**Exclude**	**random**	**learners**	**Exclude**	**regret**	**learners**	**Exclude**	**smallest**	**activities**
Exp.	**Inst.**	**ILP Objective**	**ALNS Objective**	**Run-time**	**Opt. Gap**	**ALNS Objective**	**Run-time**	**Opt. Gap**	**ALNS Objective**	**Run-time**	**Opt. Gap**	**ALNS Objective**	**Run-time**	**Opt. Gap**
1	1	1619.05	1412.01	93.03	14.66	1467.33	109.76	10.34	1502.58	130.53	7.75	1471.23	138.71	10.05
2	1	1561.15	1399.4	86.85	11.56	1423.17	153.04	9.7	1438.54	112.94	8.52	1430.73	145.84	9.12
3	1	1621.2	1388.58	136.64	16.75	1439.99	136.32	12.58	1418.4	142.92	14.3	1413.37	132.49	14.7
4	1	2589.87	2342.02	106.28	10.58	2467.93	146.55	4.94	2431.5	143	6.51	2465.02	162.51	5.06
5	1	2383.8	2164	125.78	10.16	2228.87	149.86	6.95	2208.99	167.57	7.91	2193.99	147.4	8.65
6	1	2314.8	2005.64	110.27	15.41	2056.71	174.78	12.55	2094.29	155.25	10.53	2065.6	172.02	12.06
7	1	1675.02	1484.77	84.14	12.81	1533.65	134.91	9.22	1529.67	118.62	9.5	1513.85	146.34	10.65
8	1	1483.65	1377.28	106.26	7.72	1331.25	111.34	11.45	1338.87	140.02	10.81	1338.89	128.17	10.81
9	1	1648.79	1426.62	103.3	15.57	1463.53	156.83	12.66	1435.84	120.15	14.83	1445.37	123.29	14.07
10	1	2494.7	2313.26	118.96	7.84	2382.53	168.69	4.71	2356.1	152.11	5.88	2333.36	135.65	6.91
11	1	2353.99	2061.59	130.84	14.18	2174.07	134.69	8.28	2212.2	160.29	6.41	2186.45	120.04	7.66
12	1	2287.4	1927.69	118.92	18.66	2029.73	153.11	12.69	2044.37	155	11.89	2083.58	175.89	9.78
13	1	1549.63	1398.99	125.77	10.77	1389.26	104.2	11.54	1429.74	116.82	8.39	1420.21	140.97	9.11
14	1	1610.92	1405.02	116.42	14.65	1431.48	117.57	12.54	1464.58	131.68	9.99	1442.54	121	11.67
15	1	1528.37	1270.6	105.35	20.29	1351.72	123.11	13.07	1346.94	153.3	13.47	1376.81	124.04	11.01
16	1	2450.79	2128.83	100.57	15.12	2297.37	128.29	6.68	2246.17	149.34	9.11	2297.84	152.71	6.66
17	1	2336.91	2029.99	121.06	15.12	2194.67	160.5	6.48	2197	169.04	6.37	2176.14	176.23	7.39
18	1	2176.74	1829.14	134.45	19	1953.32	156.86	11.44	1916.86	172.96	13.56	1938.87	156.19	12.27
19	1	1563.45	1395.36	124.66	12.05	1455.29	115.09	7.43	1460.59	122.13	7.04	1454.17	137.24	7.51
20	1	1601.72	1390.58	112.96	15.18	1488.65	121.58	7.6	1490.39	136.69	7.47	1442.92	118.63	11.01
21	1	1528.14	1321.08	154.68	15.67	1353.71	138.62	12.88	1374.96	139.75	11.14	1334.52	128.61	14.51
22	1	2603.86	2227.21	100.63	16.91	2425.68	142.05	7.35	2414.29	144.7	7.85	2463.54	132.86	5.7
23	1	2359.98	2147.7	94.8	9.88	2212.68	178.25	6.66	2235.68	169.93	5.56	2237.26	151.96	5.48
24	1	2243.21	1899.18	115.66	18.11	1968.03	137.03	13.98	1994.51	157.48	12.47	2037.61	166.67	10.09
25	1	1640.36	1429.96	93.78	14.71	1530.48	113.23	7.18	1463.06	108.14	12.12	1534.98	138.51	6.86
26	1	1591.13	1376.43	92	15.6	1445.09	118.77	10.11	1435.65	113.09	10.83	1476.04	148.16	7.8
27	1	1564.75	1329.27	90.78	17.72	1412.86	144.8	10.75	1404.33	166	11.42	1398.09	145.03	11.92
28	1	2411.16	2151.4	97.22	12.07	2275.92	131.3	5.94	2247	139.98	7.31	2272.94	153.09	6.08
29	1	2418.66	2235.52	102.59	8.19	2234.83	168.57	8.23	2281.79	151.62	6	2213.56	176.45	9.27
30	1	2177.08	1869.68	144.9	16.44	1940.12	136.39	12.21	1911.51	154.79	13.89	1975.91	166.18	10.18
31	1	1686.96	1475.89	115.26	14.3	1559.34	114.4	8.18	1485.96	105.34	13.53	1588.97	134.38	6.17
32	1	1625.9	1410.39	161.74	15.28	1510.37	121.35	7.65	1488.81	115.19	9.21	1485.48	127.31	9.45
33	1	1559.24	1372.79	89.87	13.58	1349.59	157.68	15.53	1355.17	142.7	15.06	1368.66	104.67	13.93
34	1	2525.84	2226.33	117.03	13.45	2360.23	150.44	7.02	2377.9	145.56	6.22	2309.52	129.91	9.37
35	1	2381.52	1964.41	127.3	21.23	2214.08	185.04	7.56	2189.9	132.9	8.75	2167.66	126.95	9.87
36	1	2250.51	1851.78	141.9	21.53	1960.66	221.09	14.78	1986.68	165.46	13.28	1991.89	150.79	12.98

E Full instance set outputs

Table 9: All experiments and instances with full operator set compared to ILP results for both 500 iteration and 2000 iteration heuristic solutions (part 1).

		ILP			500 iterations			2000 iterations		
Exp.	**Inst.**	**ILP Objective**	**Run-Time**	**600s Gap(%)**	**ALNS Objective**	**Run-Time**	**Opt. Gap**	**ALNS Objective**	**Run-Time**	**Opt. Gap**
1	1	1619.05	15.34	0	1489.6	113.41	8.69	1510.52	393.55	7.18
1	2	1632.47	15.24	0	1527.83	111.11	6.84	1498.98	402.82	8.91
1	3	1649.36	15.41	0	1469.68	109.15	12.22	1522.93	475.98	8.3
2	1	1561.14	142.1	0	1406.72	110.19	10.98	1437.09	519.66	8.63
2	2	1577.43	56.92	0	1431.91	121.4	10.16	1460.65	408.91	8
2	3	1544.43	53.57	0	1424.73	109.98	8.4	1413.12	442.84	9.29
3	1	1621.2	76.64	0	1411.39	133.12	14.86	1425.14	524.17	13.76
3	2	1586.82	56.65	0	1418.04	129.03	11.9	1413.36	397.6	12.27
3	3	1635.77	65.89	0	1407.44	115.32	16.22	1469.57	439.24	11.31
4	1	2589.86	26.77	0	2462.52	139.86	5.17	2409.31	412.91	7.49
4	2	2404.46	24.98	0	2197.52	141.9	9.41	2270.54	465.88	5.9
4	3	2410.02	24.4	0	2239.88	122.36	7.59	2263.34	406.45	6.48
5	1	2383.8	313.48	0.4	2240.02	157.95	6.42	2223.27	464.73	7.22
5	2	2411.42	380.56	0	2238.55	142.28	7.72	2284.36	442.14	5.56
5	3	2284.8	510.16	0	2155.32	156.73	6	2128.31	411.5	7.35
6	1	2314.79	1092.39	0.27	2116.26	168.7	9.38	2074.63	500.36	11.58
6	2	2274.79	879.17	0.17	2072.88	149.79	9.74	2042.49	556.62	11.37
6	3	2179.45	2811.54	0.51	1984.78	155.17	9.8	2002.41	458.67	8.84
7	1	1675.01	17.58	0	1504.97	121.47	11.3	1572.85	330.68	6.5
7	2	1639.61	14.42	0	1491.02	116.16	9.96	1516.04	337.87	8.15
7	3	1656.48	15.01	0	1501.51	116.43	10.32	1534.99	411.19	7.92
8	1	1483.65	64.47	0	1311.27	97.3	13.15	1365.63	425.44	8.64
8	2	1622.48	68.56	0	1512.59	120.88	7.26	1507.78	486.39	7.61
8	3	1632.45	63.87	0	1461	111.42	11.73	1525.92	476.7	6.98
9	1	1648.79	93.85	0	1455.66	127.33	13.27	1420.14	452.38	16.1
9	2	1633.11	31.31	0	1419.23	111.49	15.07	1461.44	445.43	11.75
9	3	1609.42	82.33	0	1433.88	114.72	12.24	1440.29	574.47	11.74
10	1	2494.7	28.04	0	2326.16	125.61	7.25	2397.75	562.27	4.04
10	2	2424.82	25.11	0	2224.15	113.8	9.02	2272.98	525.24	6.68
10	3	2584.23	27.23	0	2425.94	146.47	6.52	2432.25	550.93	6.25
11	1	2353.99	725.8	0.13	2145.31	122.35	9.73	2198.47	542.45	7.07
11	2	2355.35	289.35	0	2228.92	138.07	5.67	2199.31	546.64	7.1
11	3	2310.27	1144.79	0.2	2158.54	164.29	7.02	2150.9	540.91	7.41
12	1	2287.39	996.87	0.37	2074.94	161.69	10.24	2075.54	511.95	10.21
12	2	2217.73	451.83	0	1998.24	151.6	10.98	1995.74	528.8	11.12
12	3	2230.72	648.86	0.09	1979.3	138.86	12.7	2043.76	544.72	9.15
13	1	1549.63	17.81	0	1439.05	109.89	7.68	1424.12	420.02	8.81
13	2	1589.41	14.56	0	1431.29	105.24	11.04	1461.02	398.27	8.79
13	3	1664.83	19.53	0	1441.34	97.16	15.5	1552.94	340.08	7.21
14	1	1610.91	74.64	0	1462.31	128.73	10.16	1486.39	362.09	8.38
14	2	1580.99	63.53	0	1429.76	126.28	10.57	1430.08	377.29	10.55
14	3	1576	65.8	0	1415.09	115.49	11.37	1456.17	420.15	8.23
15	1	1528.36	98.51	0	1350.08	136.5	13.21	1357.14	394.98	12.62
15	2	1591.58	52.48	0	1450.75	122.15	9.7	1415.03	415.2	12.48
15	3	1650.07	62.8	0	1428.12	94.59	15.54	1486.83	419.88	10.98
16	1	2450.78	29.84	0	2225.95	112.23	10.1	2247.1	327.31	9.06
16	2	2340.79	27.73	0	2157.75	119.85	8.48	2214.92	452.37	5.68
16	3	2439	27	0	2250.13	120.19	8.39	2281.58	424.51	6.9
17	1	2336.91	456.59	0	2192.14	133.32	6.6	2208.96	458.01	5.79
17	2	2362.35	252.95	0	2175.19	140.23	8.6	2215.25	463.59	6.64
17	3	2259.85	882.77	0.21	2102.57	136.99	7.48	2104.51	515.62	7.38
18	1	2176.74	2818.07	0.47	1965.6	129.78	10.74	1985.31	445.54	9.64
18	2	2273.79	4479.96	0.52	2033.32	130.33	11.82	2083.44	535.47	9.14
18	3	2275.15	789.4	0.19	2017.58	154.56	12.76	2013.99	542.24	12.97

Table 10: All experiments and instances with full operator set compared to ILP results for both the 500 iteration and 2000 iteration heuristic solutions (part 2).

		ILP			500 iterations			2000 iterations		
Exp.	**Inst.**	**ILP Objective**	**Run-Time**	**600s Gap(%)**	**ALNS Objective**	**Run-Time**	**Opt. Gap**	**ALNS Objective**	**Run-Time**	**Opt. Gap**
19	1	1563.44	20.62	0	1435.8	98.94	8.89	1414.26	458.44	10.55
19	2	1607.35	17.78	0	1470.94	107.05	9.27	1494.18	416.79	7.57
19	3	1674.82	18.41	0	1535.48	112.94	9.07	1569.08	436.55	6.74
20	1	1601.72	103.36	0	1486.79	118.2	7.73	1484.44	508.26	7.9
20	2	1550.41	79.14	0	1411.45	81.16	9.84	1433.62	432.25	8.15
20	3	1675.98	92.06	0	1535.75	110.05	9.13	1587.98	366.49	5.54
21	1	1528.13	73.07	0	1353.9	122.66	12.87	1379.58	408.22	10.77
21	2	1540.38	91.89	0	1360.71	129.05	13.2	1400.11	411.51	10.02
21	3	1633.57	130.48	0	1453.53	112.98	12.38	1460.2	331.99	11.87
22	1	2603.85	25.89	0	2441.52	119.72	6.65	2457.9	342.42	5.94
22	2	2407.83	29.31	0	2295.44	128.36	4.89	2249.51	337.45	7.04
22	3	2637.72	32.94	0	2492.63	125.75	5.82	2521.56	357.46	4.61
23	1	2359.97	891.83	0.18	2182.84	130.65	8.11	2232.84	406.3	5.69
23	2	2447.07	704.64	0.07	2242.61	127.25	9.11	2321.25	359.29	5.42
23	3	2288.26	1297.2	0.28	2115.49	160.09	8.16	2122.32	356.75	7.82
24	1	2243.21	1878.62	0.27	1966.07	134.11	14.1	2044.96	368.91	9.69
24	2	2448.21	3799.5	0.53	2225.71	129.85	9.99	2212.77	427.36	10.64
24	3	2406.21	1489.12	0.2	2176.66	160.72	10.54	2183.39	434.97	10.21
25	1	1640.36	18.23	0	1540.18	104.77	6.5	1531.61	290.15	7.1
25	2	1766.77	16.48	0	1603.36	99.23	10.19	1614.37	331.38	9.44
25	3	1584.18	19.8	0	1424.47	96.09	11.21	1468.03	343.29	7.91
26	1	1591.13	65.7	0	1464.45	118.29	8.65	1446.48	359.07	10
26	2	1631.59	57.24	0	1472.84	121.53	10.77	1496.75	319.62	9.01
26	3	1676.56	64.9	0	1493.45	131.03	12.26	1570.76	329.9	6.74
27	1	1564.75	89.18	0	1396.07	142.36	12.08	1431.62	405.19	9.3
27	2	1599.62	94.41	0	1414.68	165.56	13.07	1474.41	359.94	8.49
27	3	1532.21	104.66	0	1375.31	141.2	11.4	1387.42	333.49	10.44
28	1	2411.16	29.22	0	2229.06	145.87	8.17	2321.12	339.77	3.88
28	2	2563.27	32.29	0	2405.6	170.39	6.55	2428.57	359.78	5.55
28	3	2501.78	33.2	0	2360.59	197.24	5.98	2385.85	387.55	4.86
29	1	2418.66	1231.04	0.21	2291.79	188.48	5.54	2328.43	471.92	3.88
29	2	2305.68	2349.46	0.38	2169.5	184.89	6.27	2130.5	407.43	8.22
29	3	2555.21	1288.06	0.24	2378.38	198.31	7.43	2384.77	399.09	7.15
30	1	2177.08	1449.11	0.36	1956.1	164.3	11.3	1987.6	415.94	9.53
30	2	2281.73	1011.18	0.24	2019.45	164.26	12.98	2050.19	407.39	11.29
30	3	2256.52	1443.78	0.28	2031.86	176.41	11.05	2053.7	487.72	9.88
31	1	1686.95	16.21	0	1454.44	140	15.99	1546.47	299.35	9.08
31	2	1575.24	25.59	0	1436.47	131.03	9.66	1454.76	311.47	8.28
31	3	1626.5	20.36	0	1489.15	135.61	9.22	1493.71	317.08	8.89
32	1	1625.89	27.83	0	1498.28	181.98	8.52	1506.25	332.83	7.94
32	2	1487.55	76.46	0	1351.92	170.31	10.03	1371.49	321.5	8.46
32	3	1633.51	93.59	0	1481.03	156.23	10.29	1510.49	325.89	8.14
33	1	1559.24	77.48	0	1358	127.36	14.82	1397.31	371.15	11.59
33	2	1545.51	68.05	0	1387.73	173.43	11.36	1435.38	466.39	7.67
33	3	1572.13	80.74	0	1422.01	144.29	10.55	1420.54	417.68	10.67
34	1	2525.83	26.79	0	2349.34	142.95	7.51	2412	359.34	4.72
34	2	2459.06	30.52	0	2267.67	171.16	8.44	2278.27	334.77	7.94
34	3	2572.24	29.67	0	2342.88	132.03	9.78	2443.91	368.27	5.25
35	1	2381.52	332.16	0	2251.78	178.03	5.76	2212.68	355.55	7.63
35	2	2403.9	937.16	0.13	2266.15	159.44	6.07	2262.07	400.99	6.27
35	3	2335.02	684.94	0.06	2202.16	141.44	6.03	2190.53	403.33	6.6
36	1	2250.5	1006.23	0.22	2032.58	156.45	10.72	2037.19	371.77	10.47
36	2	2220.96	2349.92	0.47	1982.87	148.32	12	1997.14	410.82	11.21
36	3	2416.09	8805.4	0.71	2113.88	145.77	14.29	2136.07	396.75	13.11