See the Slots - Book the Spots; Optimizing a Vaccination Scheduling Campaign

Athanasios Tsiamis
Faculty of Science, Utrecht University, The Netherlands
a.tsiamis@students.uu.nl

Julian Markus
Faculty of Science, Utrecht University, The Netherlands
j.a.w.markus@students.uu.nl

Leona Teunissen
Faculty of Science, Utrecht University, The Netherlands
l.j.teunissen@students.uu.nl

Morice Ebbertz
Faculty of Science, Utrecht University, The Netherlands
m.ebbertz@students.uu.nl

Ramón R. Cuevas
Faculty of Science, Utrecht University, The Netherlands
r.ricocuevas@student.uu.nl

Abstract

This paper aims to study a scheduling problem called the vaccine scheduling problem. The objective of the paper is to provide exact solutions for small instances in the offline setting and a general strategy to deal with the online setting. We propose an algorithm based on ILP modeling for the offline setting and an algorithm based on a best-fit heuristic for the online setting. For the latter one, we prove a competitive ratio lower bound of $\lg(n)$ where n is the number of patients. Furthermore, we conduct a series of experiments to test the performance of our proposed algorithms using the test instances provided by fellow Utrecht University students and some randomly generated instances. As the experiments show, our offline algorithm is able to deal with small instances. For future research, we aim at improving the offline algorithm in order to be able to deal with larger instances as well as providing an upper bound for the competitive ratio of our proposed online algorithm.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Offline Algorithms, Online Algorithms, Scheduling, Vaccine Scheduling

Supplementary Material https://git.science.uu.nl/ThanosTsiamis/algorithms-for-decision-support-assignment-1

1 Introduction

In this first section of the paper, the vaccine scheduling problem is formally introduced. Similar classic scheduling problems are examined and compared against the problem at hand.

1.1 The vaccine scheduling problem

Suppose a very contagious unnamed disease has spread in an unnamed country. The national regulatory authorities want to vaccinate a portion of the population of the country in order to reach herd immunity. There are $n \in \mathbb{N}$ citizens eligible to be vaccinated and we will refer to them as patients. The vaccination is performed using two-phase vaccine jabs. Each vaccine jab has two doses and a minimum time gap $g \in \mathbb{Z}_+$ is required between the two doses. The doses must be administered in a hospital that is suitable for vaccination purposes.

Each hospital has a number of time slots in which patients can be vaccinated at and can only attend one patient per time slot. Once a patient gets the first dose at a hospital, he/she must remain under observation at the same hospital for a certain number of contiguous time slots. This number is known as the processing time of the first dose and will be denoted by p_1. The same is true about the second dose, and in this case will be denoted by p_2. The processing times $p_1, p_2 \in \mathbb{N}$ and are provided by the pharmaceutical company in charge of manufacturing the vaccine. It is important to note that although a patient must remain in the same hospital during the processing time of either dose, it is not necessary to administer both doses in the same hospital. Each patient $P_i,\ i = 1, \ldots, n$ is asked to submit via the government's health services web page a list of four numbers:

$$(r_{i,1},\, d_{i,1},\, \alpha_i,\, l_i) \in \mathbb{N} \times \mathbb{N} \times \mathbb{Z}_+ \times \mathbb{N}$$

where,

- $r_{i,1}$ is the lower bound of the first feasible interval $I_{i,1}$ (i.e. the time interval during which he/she is available to get the first dose).
- $d_{i,1} (\geq r_{i,1})$ is the upper bound of the first feasible interval $I_{i,1}$.
- α_i is the patient - dependant delay (i.e. the number of time slots he/she would like to wait between the first and second dose in addition to the mandatory time gap g).
- $l_i (\geq p_2)$ is length of the second feasible interval $I_{i,2}$.

Given this information, the system must send back to each patient a list of four numbers:

$$(t_{i,1},\, H_{i,1},\, t_{i,2},\, H_{i,2}) \in \mathbb{N}^4$$

where,

- $t_{i,1}$ is the time slot when patient P_i will get the first dose.
- $H_{i,1}$ is the hospital number where patient P_i will get the first dose.
- $t_{i,2}$ is the time slot when patient P_i will get the second dose.
- $H_{i,2}$ is the hospital number where patient P_i will get the second dose.

Each of these numbers are calculated in the following way. The first dose is scheduled at start time $t_{i,1} \in I_{i,1} = [r_{i,1},\, d_{i,1}]$ in an available hospital $H_{i,1}$ such that $[t_{i,1},\, t_{i,1} + p_1 - 1] \subseteq I_{i,1}$. Once the first dose is scheduled, the second feasible interval is calculated as follows

$$I_{i,2} = [t_{i,1} + p_1 + g + \alpha_i,\, t_{i,1} + p_1 + g + \alpha_i + l_i - 1].$$

The second dose is then scheduled at start time $t_{i,2} \in I_{i,2}$ in an available hospital $H_{i,2}$ such that $[t_{i,2},\, t_{i,2} + p_2 - 1] \subseteq I_{i,2}$.

The objective is to vaccinate all the patients using as less hospitals as possible. We will consider two variants of the problem.

Variant 1

In the first variant of the problem, the global parameters p_1, p_2, g and the set of jobs

$$\{(r_{1,1},\, d_{1,1},\, \alpha_1,\, l_1), \ldots, (r_{n,1},\, d_{n,1},\, \alpha_n,\, l_n)\}$$

are given beforehand. With all of this information, the system must be able to elaborate and send to each patient a list of four numbers as we explained before. We will later refer to this variant as the offline problem (see section 2).

Variant 2

In the second variant of the problem, the global parameters p_1, p_2, g are given beforehand but not the jobs. In this case there are n consecutive rounds (one for each patient). At round i, we obtain patient P_i's information, i.e.

$$(r_{i,1},\, d_{i,1},\, \alpha_i,\, l_i).$$

The program then has to schedule patient P_i, i.e. give the time $t_{i,1}$ and hospital $H_{i,1}$ when and where the first dose is given, the time $t_{i,2}$ and hospital $H_{i,2}$ when and where the second dose is given fulfilling the conditions explained earlier. After this, the next round starts with the next patient. We will later refer to this variant as the online problem (see section 2).

Research in scheduling problems has been around for a long time and particularly active in the past decades. A classic scheduling problem is the bin packing problem (see Garey et al [4]). In the bin packing problem, a series of items with sizes less than or equal to one are given. The goal is to minimize the amount of one capacity bins needed to pack all of the different-size items. For us, the bin packing problem was a staring point on thinking about a solution for the vaccine scheduling problem. At the beginning, we thought that the problems were similar but then we realised that there are some very big differences between the two. In the bin packing problem, the capacity of the bins is finite whereas in the vaccine scheduling problem hospitals are assumed to have infinite time slots. Also, in the bin packing problem the items are not required to have the same size and no interval constraints are imposed. Therefore, we started looking at literature related to scheduling problems with interval constraints and no capacity limitations. We then realised that the vaccine scheduling problem is complex variant of the machine minimization problem (see Chuzhoy et al. [1]). In this problem, a number of jobs ,that need to be scheduled in a certain number of machines, is given. Each one of these jobs has a feasible interval inside of which must be completed. Also, each machine can only process one job at a time. The goal is to minimize the amount of machines needed for carrying out this scheduling task. In the case where all of the information with respect to the jobs is given beforehand, the problem is solved via linear programming (see definition 10 in section 2). This gave us the inspiration that variant 1 of the vaccine scheduling problem could be solved using the same technique. In the case where jobs are revealed sequentially (see Devanur et al. [2]) an algorithm based on a heuristic criteria is used to solve the problem. This gave us the idea of using a similar technique for solving variant 2 of the vaccine scheduling problem. In this paper we propose solutions for both variants of the vaccine scheduling problem using as a starting point the problems described above.

2 Preliminaries

In this section we introduce a series of definitions that will make it easier to contextualize the vaccine scheduling problem and will lay the ground for better understanding of the following sections.

▶ **Definition 1.** *An* ***optimization problem*** Π *consists of a set of* ***instances*** *or* ***jobs*** $\mathcal{J}$*, a set of* ***feasible solutions*** $\mathcal{O}$*, and a* ***cost function***

$$cost : \mathcal{O} \longmapsto \mathbb{R}.$$

Every instance $J \in \mathcal{J}$ *is a sequence of requests* $J = (x_1, x_2, \ldots, x_n)$ *and every feasible solution* $O \in \mathcal{O}_J \subset \mathcal{O}$ *is a sequence of answers* $O = (y_1, y_2, \ldots, y_n)$*, where* $n \in \mathbb{N}$*. Note* *that $\mathcal{O} = \bigcup_{J \in \mathcal{J}} \mathcal{O}_J$. Given an instance J and a corresponding feasible solution $O \in \mathcal{O}_J$, the* ***cost*** *associated with solution O is denoted by $cost(O)$. Whether the goal is to minimize or maximize the cost function, optimization problems can be further divided into* ***minimization*** *and* ***maximization*** *problems.*

▶ **Definition 2.** *An* ***optimal solution*** *for an instance $J \in \mathcal{J}$ of a* ***minimization*** *(optimization) problem Π as in 1 is a solution $OPT(J) \in \mathcal{O}_J$ such that,*

$$cost(OPT(J)) = \min_{O \in \mathcal{O}_J} cost(O).$$

i.e., an optimal solution for a minimization problem is a feasible solution that obtains the minimum cost.

▶ **Definition 3.** *An* ***optimal solution*** *for an instance $J \in \mathcal{J}$ of a* ***maximization*** *(optimization) problem Π as in 1 is a solution $OPT(J) \in \mathcal{O}_J$ such that,*

$$cost(OPT(J)) = \max_{O \in \mathcal{O}_J} cost(O).$$

i.e., an optimal solution for a maximization problem is a feasible solution that obtains the maximum cost.

▶ **Definition 4.** *An* ***offline problem*** *is an optimization problem Π as in 1 such that the set of instances $\mathcal{J}$ is available all at once.*

▶ **Definition 5.** *An* ***online problem*** *is an optimization problem Π as in 1 such that the input instances $J \in \mathcal{J}$ are revealed sequentially.*

▶ **Definition 6.** *An* ***offline algorithm*** *is a rule to solve an offline problem Π as in 4. Note that due to the nature of offline problems, an offline algorithm is allowed to consider the entire set of instances $\mathcal{J}$ to compute the optimal solution of problem Π.*

▶ **Definition 7.** *An* ***online algorithm*** *is a rule to solve an online problem Π as in 5. Note that due to the nature of online problems, an online algorithm must make a decision upon the arrival of each request $J \in \mathcal{J}$ without knowledge about the future. Moreover, the decisions are irrevocable. That is, the decisions are permanent and cannot be changed afterwards.*

▶ **Definition 8.** *Consider a minimization online problem Π. An online algorithm ALG is* ***c-competitive*** *if*

$$\exists\, \alpha \in \mathbb{R} \;:\; \forall\, J \in \mathcal{J}, \quad cost(ALG(J)) \leq c \cdot cost(OPT(J)) + \alpha.$$

i.e., there exists a constant α such that for every finite instance $J \in \mathcal{J}$ the cost incurred by the online algorithm ALG is bounded by c times the cost incurred by the optimal solution.

▶ **Definition 9.** *Consider a minimization online problem Π and an online algorithm ALG. If there exists an instance J such that*

$$\frac{cost(ALG(J))}{cost(OPT(J))} \geq l$$

for some constant $l \in \mathbb{R}$, by definition 8 we know that, ALG cannot be c-competitive for any $c < l$. We call the constant $l \in \mathbb{R}$ a ***competitive ratio lower bound*** *of the online algorithm ALG.*

▶ **Definition 10.** *Consider a minimization offline problem* Π*. The* ***linear programming*** *formulation or LP formulation of problem* Π *is,*

$$min \sum_{i=1}^{n} c_i x_i,$$

subject to

$$\sum_{i=1}^{n} a_{i1} x_i \leq b_1,$$

$$\vdots$$

$$\sum_{i=1}^{n} a_{im} x_i \leq b_m,$$

$$x_i \geq 0, \quad \forall\, i \in \{1, \ldots, n\}.$$

The LP formulation of offline minimization problem Π *is a way of writing down the problem such that the solution is encoded by* $n \in \mathbb{N}$ *variables* $x_1, \ldots, x_n$ *called* ***decision variables*** *with associated costs* $c_1, \ldots, c_n$ *and the objective is to minimize the total cost. Therefore, the* ***objective function*** *is given by the expression* $min \sum_{i=1}^{n} c_i x_i$*. The* n *decision variables are subject to* $m \in \mathbb{N}$ ***constraints*** *of the form* $\sum_{i=1}^{n} a_{ij} x_i \leq b_j$*, where* $a_{ij}, b_j \in \mathbb{R}$*; as well as* n *domain constraints,* $x_i \geq 0$*. An* ***optimal solution*** *in this context is any solution that satisfies all the constraints and achieves minimal cost.*

▶ **Definition 11.** *Consider a minimization offline problem* Π*. The* ***integer linear programming*** *formulation or ILP formulation of problem* Π *is,*

$$min \sum_{i=1}^{n} c_i x_i,$$

subject to

$$\sum_{i=1}^{n} a_{i1} x_i \leq b_1,$$

$$\vdots$$

$$\sum_{i=1}^{n} a_{im} x_i \leq b_m,$$

$$x_i \in \mathbb{Z}_+, \quad \forall\, i \in \{1, \ldots, n\}.$$

Note that the ILP formulation of offline minimization problem Π *only differs from the LP formulation in the* n *domain constraints. In the case of ILP the decision variables* x_i *are forced to be non negative integers.*

It is important to note that linear programs are very efficiently solvable. In the other hand, integer linear programs are not. Nevertheless, there are techniques of solving integer linear programs like branch and bound. A major advantage of modeling a given problem as an LP or ILP is that there exist many available solvers. Therefore, given a minimization offline problem Π, building an offline algorithm ALG to solve the problem could be achieved by giving an LP or ILP formulation of Π. The algorithm ALG would be described by the LP or ILP formulation plus a state-of-the-art solver.

All the concepts introduced in this section now allow us to better contextualize the vaccine scheduling problem described in section 1. In fact, the vaccine scheduling problem is a minimization problem that comes in two flavours. variant 1 of the problem is an offline minimization problem, and variant 2 of the problem is an online minimization problem. In the following sections we aim at giving two algorithms, one to solve the offline version of vaccine scheduling and one to solve the online version of vaccine scheduling.

3 Proposed solution for the offline setting

This section is dedicated to the offline version of the vaccine scheduling problem. As we discussed in section 2, modeling our problem as an ILP would be enough in the offline setting to obtain an offline algorithm.

3.1 Summary of the problem

In the first place, we briefly summarize the problem described in Section 1 and create appropriate parameters and decision variables to formulate the ILP.

Data

- n patients to be vaccinated.
- n potential hospitals where patients could be vaccinated at.
- T time intervals per hospital on which patients could be attended on.
- Each patient must get two doses.
- Each hospital can only process 1 patient per time slot.

Global parameters

- Processing time of the first dose $p_1 \geq 1$.
- Processing time of the second dose $p_2 \geq 1$.
- Mandatory time gap between the first and the second doses g.

Patient-dependent parameters

- The patient-dependent lower bound of the first dose feasible interval $r_{i,1}$.
- The patient-dependent upper bound of the first dose feasible interval $d_{i,1}$.
- The patient-dependent delay α_i where $\alpha_i \geq 0$.
- The patient-dependent (second dose) feasible interval length l_i where $l_i \geq p_2$.

With this information, we define the patient-dependant parameter a_{it} that models if patient P_i, $i \in \{1, \ldots, n\}$ **CAN GET** the first dose at time slot $t \in \{1, \ldots, T\}$.

$$a_{it} := \begin{cases} 1 & \text{if} \quad r_{i,1} \leq t \leq d_{i,1} - p_1 + 1, \\ 0 & \text{otherwise.} \end{cases}$$

3.2 ILP Formulation

Decision variables

We define the following decision variables,

$$x_j = \begin{cases} 1 & \text{if hospital } H_j \textbf{ IS} \text{ used for vaccination purposes,} \\ 0 & \text{otherwise.} \end{cases}$$

$$y_{itj} = \begin{cases} 1 & \text{if patient } P_i \textbf{ GETS} \text{ first dose at time } t \text{ in hospital } H_j, \\ 0 & \text{otherwise.} \end{cases}$$

$$z_{itj} = \begin{cases} 1 & \text{if patient } i \textbf{ GETS} \text{ second dose at time } t \text{ in hospital } H_j, \\ 0 & \text{otherwise.} \end{cases}$$

Here, $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, n\}$ and $t \in \{1, \ldots, T\}$.

Objective function

The objective is the minimization of the number of hospitals needed to carry out the vaccination. Therefore the objective function becomes

$$\min \sum_{j=1}^{n} x_j.$$

Constraints

$$\sum_{t=1}^{T} \sum_{j=1}^{n} y_{itj} = 1, \quad \forall\, i \in \{1 \ldots, n\}, \tag{1}$$

$$\sum_{t=1}^{T} \sum_{j=1}^{n} z_{itj} = 1, \quad \forall\, i \in \{1 \ldots, n\}, \tag{2}$$

$$\sum_{i=1}^{n} \left(\sum_{k=t-p_1+1}^{t} y_{ikj} + \sum_{k=t-p_2+1}^{t} z_{ikj} \right) \leq 1, \quad \forall\, t \in \{1, \ldots, T\},\ j \in \{1 \ldots, n\}, \tag{3}$$

$$y_{itj} \leq a_{it} x_j, \quad \forall\, i \in \{1 \ldots, n\},\ t \in \{1, \ldots, T\},\ j \in \{1 \ldots, n\}, \tag{4}$$

$$\sum_{j=1}^{n} z_{itj} \leq \sum_{j=1}^{n} \left(\sum_{k=t-p_1-g-\alpha_i-l_i+p_2}^{t-p_1-g-\alpha_i} y_{ikj} \right), \quad \forall\, t \in \{1, \ldots, T\},\ i \in \{1, \ldots, n\}, \tag{5}$$

$$z_{itj} \leq x_j, \quad \forall\, i \in \{1 \ldots, n\},\ t \in \{1, \ldots, T\},\ j \in \{1 \ldots, n\}, \tag{6}$$

$$x_{j+1} \leq x_j, \quad \forall\, j \in \{1 \ldots, n-1\}, \tag{7}$$

$$x_j,\ y_{itj},\ z_{itj} \in \{0, 1\}, \quad \forall\, i \in \{1 \ldots, n\},\ t \in \{1, \ldots, T\},\ j \in \{1 \ldots, n\}. \tag{8}$$

Description of the constraints

(1) Each patient gets the first dose of the vaccine exactly once. For each patient P_i we add up the decision variables y_{itj} over all time slots and over all hospitals. This sum has to be equal to one in order to make the desired condition hold.

(2) Each patient gets the second dose of the vaccine exactly once. For each patient P_i we add up the decision variables z_{itj} over all time slots and over all hospitals. This sum has to be equal to one in order to make the desired condition hold.

(3) Each hospital can only process one patient at a time. For each hospital H_j and for each time slot t we add up the decision variables y_{ikj} and z_{ikj} summing over all patients and over all time slots $k \in \{t-p_1+1,\dots,t\}$ and $k \in \{t-p_2+1,\dots,t\}$ respectively. If patient P_i gets the first dose at hospital H_j then $y_{it_{i,1}j} = 1$. But recall that he/she must remain in hospital H_j for p_1 contiguous time slots, i.e. hospital H_j should have time interval $[t_{i,1}, t_{i,1}+p_1]$ reserved for patient P_i. Similarly, if patient P_i gets the second dose at hospital H_j then $z_{it_{i,2}j} = 1$. But recall that he/she must remain in hospital H_j for p_2 contiguous time slots, i.e. hospital H_j should have time interval $[t_{i,2}, t_{i,2}+p_2]$ reserved for patient P_i. In this case, the following two conditions hold

$$\sum_{k=t-p_1+1}^{t} y_{ikj} = 1, \quad \forall\, t \in [t_{i,1}, t_{i,1}+p_1],$$

$$\sum_{k=t-p_2+1}^{t} z_{ikj} = 1, \quad \forall\, t \in [t_{i,2}, t_{i,2}+p_2].$$

Therefore, if we want $[t_{i,1}, t_{i,1}+p_1] \cap [t_{i,2}, t_{i,2}+p_2] = \emptyset$ we have to impose,

$$\sum_{k=t-p_1+1}^{t} y_{ikj} + \sum_{k=t-p_2+1}^{t} z_{ikj} \leq 1, \quad \forall\, t \in \{1,\dots,T\}.$$

To take into account the information from all patients and ensure that

$$[t_{i,m}, t_{i,m}+p_m] \cap [t_{i,n}, t_{i,n}+p_n] = \emptyset$$

(i.e. any 2 intervals chosen are disjoint) for every choice $n, m \in \{1,2\},\ i \in \{1,\dots,n\}$, it suffices to sum over all patients. This way we obtain the desired equation

$$\sum_{i=1}^{n} \left(\sum_{k=t-p_1+1}^{t} y_{ikj} + \sum_{k=t-p_2+1}^{t} z_{ikj} \right) \leq 1, \quad \forall t \in \{1,\dots,T\}.$$

To take into account all of the hospitals, it suffices to consider the above equation $\forall\, j \in \{1\dots,n\}$.

(4) A patient can only get the first dose when he is available and in an existing hospital.

(5) A patient can only get the second dose when he is available and when he already has received the first dose. Constraint (5) is built based on two observations. First, note that by summing over all hospitals we merge the n discrete timelines (one for each hospital)

into a single timeline. Second, note that if patient P_i is first-dose vaccinated at time $t_{i,1}$ then

$$\sum_{k=t-p_1-g-\alpha_i-l_i+p_2}^{t-p_1-g-\alpha_i} y_{ikj} = 1, \quad \forall\, t \in [t_{i,1}+p_1+g+\alpha_i,\, t_{i,1}+p_1+g+\alpha_i+l_i-p_2] \subseteq I_{i,2}.$$

Combining both observations we obtain equation (5).

$$\sum_{j=1}^{n} z_{itj} \leq \sum_{j=1}^{n} \left(\sum_{k=t-p_1-g-\alpha_i-l_i+p_2}^{t-p_1-g-\alpha_i} y_{ikj} \right).$$

Looking at a single discrete timeline, equation (5) imposes that the possible time intervals t where the decision variables z_{itj} could take the value 1 are precisely those time slots $t \in I_{i,2}$ such that $[t, t+p_2] \subseteq I_{i,2}$. Note also that by construction, equation (5) forces every patient to be first-dose vaccinated before being second-dose vaccinated.

(6) A patient can only get the second dose in an existing hospital. Note that both equations (6) and (5) are needed to impose for the second dose the same constraint as (4) alone imposes for the first dose. This has to do with the fact that availability for the first dose is a parameter while availability for the second dose is first-dose dependant and therefore a variable. Thus, a parameter for availability for the second dose cannot be defined from the data.

(7) Hospital H_j must be used before using H_{j+1}. The implementation chosen assumes that there exists as many hospitals as patients and minimizes the number of hospitals used for vaccination purposes. In order to "change" the status of a hospital from "regular" to "used for vaccination" in increasing order we must add this constraint. This way we avoid outputs like: The set of patients can all get vaccinated at a the single hospital H_3. In this case the program should return something like: The set of patients can all get vaccinated at a the single hospital H_1.

(8) Integrality.

4 Proposed solution for the online setting

This section is dedicated to the online version of the vaccine scheduling problem. As we discussed in section 2, we propose an algorithm based on a heuristic. Due to the nature of online machine minimization scheduling problems it is not guaranteed that our algorithm is able to make the optimal choice when scheduling a patient. E.g. when a patient gets planned, the subsequent patient may conflict (partially) with the existing planning due to an arbitrary planning decision that the algorithm made. Instead, we may be able to minimise the chance of a new patient conflicting with the existing schedule by planning the patients according to a certain heuristic. This will in turn reduce the need to open new hospitals.

4.1 Proposed heuristic

Diepen et al. [3] present an optimization algorithm for creating a gate planning. They successfully use a cost heuristic to increase robustness of the schedule to account for potential delays occurring during operations. We will construct a heuristic in the same spirit, which will attempt to maximise the size of 'free intervals' in the schedule.

Let us define a free interval to be a uninterrupted sequence of time slots in which no patient has been assigned. We reason that a larger free interval has a higher probability to accommodate a new patient. Let flexibility be a measure based on a set of free intervals I, noted as $f(I)$. We use the following function to compute the flexibility of I:

$$f(I) = \sum_{[i,j]\in I} tan^{-1}(j-i) \tag{9}$$

We remark the following desirable properties of the flexibility function:

▶ Remark 12. Larger intervals have a higher flexibility: E.g. $f([0,10]) > f([0,9])$.

▶ Remark 13. Flexibility of intervals can be summed to compare their total flexibility: E.g. $f(\{[0,4],[6,10]\}) = f([0,4]) + f([6,10])$

▶ Remark 14. A change in the interval length is weighted heavier for a small interval compared to a large interval: E.g. $f([0,3]) - f([0,2]) > f([0,10]) - f([0,9])$

Some additional edge cases exist for which flexibility needs to be determined separately:

1. Free intervals may appear of a length smaller than the lowest vaccine duration.
2. A patient may be able to be planned directly after another patient.

The first edge case should be avoided as leaving a free interval in which no future patients can be scheduled is inefficient, potentially resulting in the final planning requiring more hospitals. To avoid this, we add the condition that any interval $[i,j]$ for which $j - i < \min(p_1, p_2)$ will have $f([i,j]) = 0$. The second edge case is preferred as planning the patient at the start or end of a free interval avoids splitting the interval into two smaller intervals that are less flexible individually. Therefore, when a permutation of intervals is evaluated in which a dose is planned seamlessly at the start or end of an interval an additional constant $\alpha > \frac{\pi}{2}$ is added to the flexibility score. Choosing $\alpha = \frac{\pi}{2}$ ensures that the feasible options where a patient doesn't split existing intervals yield higher flexibility scores than options where patients do split intervals, even when interval lengths approach infinity. The modified equation that includes the conditional constant α is given by the following expression.

$$f(I) = \sum_{[i,j]\in I} tan^{-1}(j-i) + \alpha \tag{10}$$

4.2 Planning Algorithm

We propose two algorithms featuring the flexibility heuristic as mentioned in section 4.1. Listing 1 in appendix 7 shows the high level execution of a sequential planning algorithm. The planning procedure receives a set S containing free intervals per machine, as well as information about the patient's availability and proceeds to find the time slot best suited to administer the first vaccine dose. After this time slot has been determined, it is planned into the schedule and won't be altered. Then the second dose is planned based on the feasible interval dependent on the time slot the first dose is planned. If either the first or second dose cannot be planned due to conflicts in the existing schedule the planning of the specific dose will be repeated with a new machine added to S, the new machine features a completely empty schedule and is therefore guaranteed to be able to accommodate the dose(s) in question.

Listing 2 shows the execution of an integrated version of the planning algorithm. The distinct difference compared to online algorithm 1 is that the first and second dose are planned at the same time. Due to this it is assumed that a more optimal remaining flexibility of the schedule is achieved, compared to Listing 1. If our heuristic is sound, this should in turn lead to fewer machines being required when both algorithms schedule an identical instance.

4.3 Bound on competitiveness

We present a lower bound on the competitive ratio by means of a worst-case adversarial input. Consider the following scheduling problem where we need to plan only one vaccine per patient, with a vaccination time of p_1. We define a (finite) list heuristic input $J = \{j_1, \dots, j_n\}$ with n jobs. We set $T = n(p_1 + \epsilon)$ where T is the latest time slot that the algorithm will consider to plan a job. Every job $j_i \in J$, has a deadline $d_i = T$. The release time for each job is dependent on its position in the list. If we let r_i be the release time of job j_i then, $r_i = p_1(i-1) + \epsilon\, i$ where $0 < \epsilon < p_1$, for all $j_i \in J$.

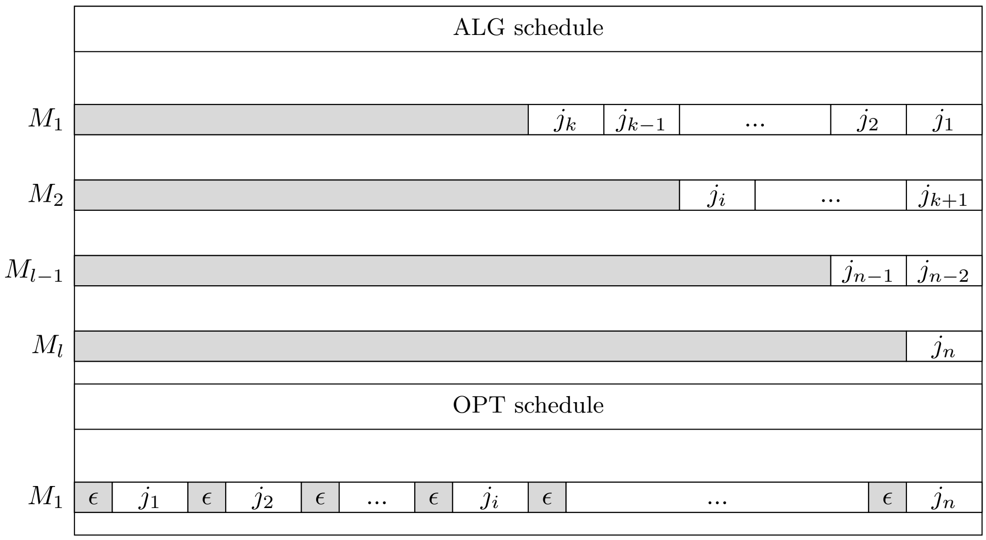

Figure 1 Comparison worst-case ALG and OPT performance

The algorithm will, based on the flexibility heuristic mentioned in the previous section, prefer to plan time slots that seamlessly connect to each other (i.e no idle time slots). Given the list heuristic starting with job j_1, this job will be planned towards the end of the schedule, as d_1 forms a seamless connection with the last possible time slot T. From this we can plan subsequent jobs, placing them back to front in the schedule without creating any new gaps.

▶ **Lemma 15.** *For two jobs j_a and j_b on the same machine M_m, job j_a will be planned later than j_b, given $b < a$.*

Proof. Follows from the list heuristic. ◀

▶ **Lemma 16.** *Per machine, subsequent jobs are ordered in non-ascending order of their release time.*

Proof. With t_b being the time slot that j_b is planned on machine M_m job j_a can be planned on M_m in the interval $[r_a, t_b - p_1)$ where $b < a$. Follows from Lemma 15. ◀

The algorithm can continue to repeat this step, until it encounters a job with a deadline later than the time slot the latest job has been planned. When encountering this job the algorithm can take no other action than to plan the current job on a new machine. The algorithm will continue prepending jobs to this machine until it encounters a new job with conflicting release time.

▶ **Lemma 17.** *A new machine M_{m+1} is introduced when a job j_a has a later release time than time slot of the first scheduled job on M_m.*

Proof. Follows from algorithm operation. ◀

▶ **Lemma 18.** *The first job scheduled on M_b has a later release time than the first job on M_a where $a < b$.*

Proof. When a new machine is introduced this implies that the new job j_i has a later release time than the planned time for the previous job, this follows from Lemma 17. Lemma 16 ensures that subsequently planned jobs feature later release times than j_i. ◀

▶ **Lemma 19.** *M_l has a single job j_n scheduled.*

Proof. The last job j_n can only be planned in the interval $[(n-1) \times p_1 + \epsilon n, n(p_1 + \epsilon)] = [T - p_1, T]$, following from the list heuristic. Due to the algorithm this interval will already be occupied on all previous machines, therefore j_n will be assigned to a new machine. ◀

We argue that this is the worst possible assignment our algorithm can make based on the following exchange argument. Suppose we pick machine M_c and M_l where $c \neq l$. The first job j_a scheduled in M_c has a later release time than the last job j_b on M_c, following from Lemma 16. The only job j_n scheduled in M_l has the latest release time possible: $T - p_1$. Due to $r_a > r_b$ we could schedule j_b *before* j_a on M_c. This now creates a vacant interval $[T - p_1, T]$ on M_c. By now placing j_n on M_c we have reduced the number of machines in the solution by 1, making the solution closer to optimal. When ϵ approaches 0 we get the following distribution of jobs per machine: $M_1 = \frac{1}{2}$ of all jobs, $M_2 = \frac{1}{2^2}$, $M_m = \frac{1}{2^m}$, etc. The last machine contains only 1 job: j_n accounting for $\frac{1}{n} = \frac{1}{2^{\lg(n)}}$ jobs, we can therefore derive the number of machines l as $l = \lg(n)$. This in turn proves a lower bound on the competitive ratio of $\lg(n)$.

We can expand the input to schedule a second dose with duration p_2 using the following input: We redefine $T_1 = 2k(p_1 + \epsilon)$, $T_2 = 2k(p_2 + \epsilon)$, and $T = T_1 + T_2$. Each job j_i has a release time $r_i = p_1(i-1) + \epsilon\, i$, first deadline $r_i = T_1$, a pause $\ell_i = T_2$ and second interval $t_i = 1$. Since the scheduling of the second dose is dependent on the first dose the algorithm will proceed to plan jobs the same as the single dose instance. The second dose's interval allows for only one time slot to be planned, making it fully dependent on the decision of how to plan the first dose. This causes no additional scheduling conflicts that are exclusive to planning the second dose, and therefore no additional machines are required compared to the single dose instance. The lower bound on the competitive ratio for this two-dose instance is therefore $\lg(n)$ as well.

5 Experimental results

In this section we will explain some experiments that we conducted in order to test our proposed online and offline algorithms.

5.1 Technical specifications

The programming language of choice in order to implement the online and the offline algorithms was Python. The code was run on Windows 10. More specifically,

- The offline algorithm was run on an 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz - 2.42 GHz. Memory utilization varied and it exceeded 16 GBs for some cases.
- The online algorithm was run on a Ryzen 9 5900X@4.40GHz with an average 6% CPU utilization. Memory utilization did not exceed 150MB.

It is important to note that for the offline algorithm program we reproduced the ILP formulation of the problem and then used an ILP solver from $OR - Tools$, an open source software suite for optimization made by Google engineers.

5.2 Offline Algorithm

To see how well our proposed offline algorithm performed we tested it against some of the test instances submitted by fellow Utrecht University students as well as against some test cases generated by a random test case function we implemented.

5.2.1 Submitted test instances

The test cases submitted by our fellow Utrecht University students vary in the amount of patients n that have to be scheduled with the smallest one having to serve 0 patients and the biggest one having to serve a million patients. The following graphs show for some of the test instances the results that our offline algorithm achieves as well as the time taken to reach the solution.

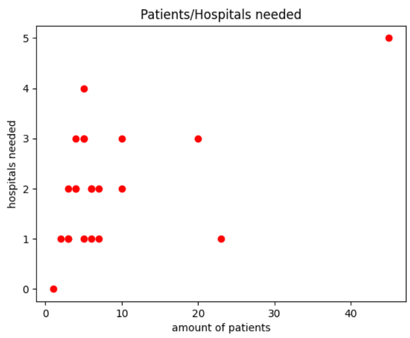

Figure 2 Hospitals used in various test cases instances

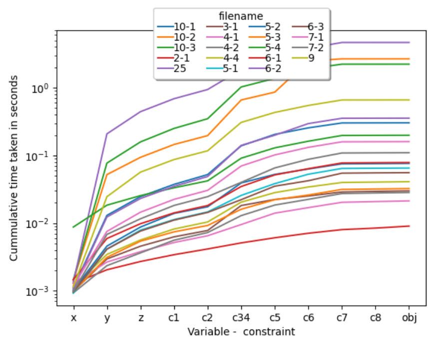

Figure 3 Time taken after computing each variable/constraint[a]

[a] Test case with 0 patients is not displayed in the figure as the time it takes is less than the process can actually measure

In Figure 2 the logarithmic scale was used to show the difference in order of magnitude between some simpler cases (e.g.one with 2 patients such as 2-1 and 10 patients such as 10-3).

We found out that for our proposed offline algorithm there is a cutoff at $n = 50$ on the amount of memory we can use on the computer. Figure 4 shows the amount of memory our program uses when running the test cases submitted by our fellow Utrecht University students. We omitted the files where the RAM exceeded 16 GBs as it crashed our computer.

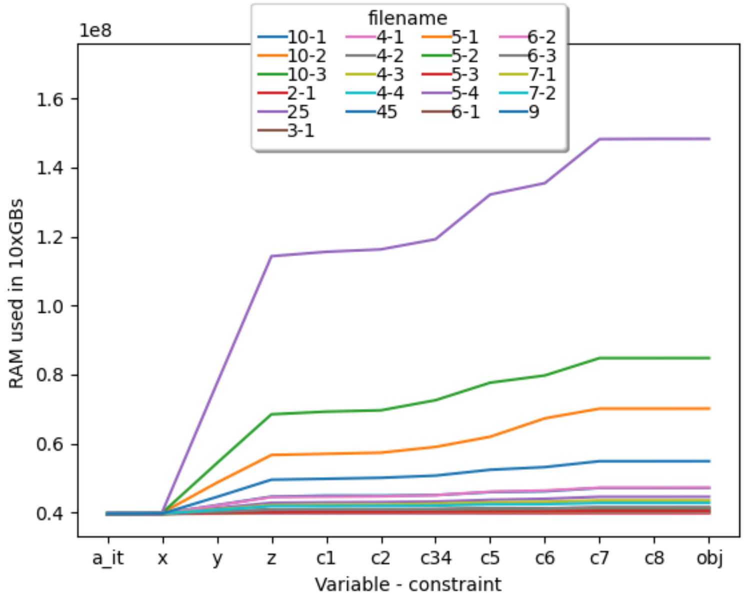

Figure 4 Memory Consumption of OR-Tools after each variable assignment for different test cases

It is clear from the figure that the decision variables x_j, y_{itj} and z_{itj} impose a heavy burden in the space complexity of the ILP increasing the memory needed by ten times.[1]

5.2.2 Random-generated test instances

To further test our offline algorithm, we developed a random test case instance generator function. This function takes as an input the variables of the ILP that we want to keep fixed (e.g. p_1, p_2 , g and/or n) and produces random values taken from the uniform distribution for the patient dependant parameters. We try to keep the values relatively small because as we found out testing our program against the submitted instances there is a cutoff at $n = 50$ patients on the amount of memory we can use on our computer. In order to see the algorithm's performance on the random generated test instances, we decided to run the program repetitively 31 times increasing by 1 the number of patients each time. The obtained results are shown in the figures below.

[1] Test case with 0 patients is not displayed in the figure as the memory it takes is less than the process can actually measure

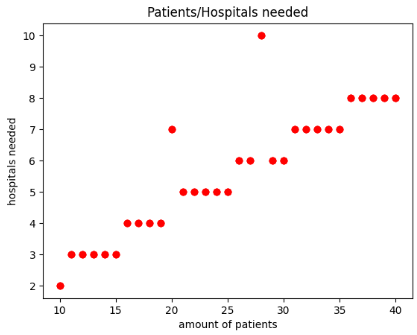

Figure 5 Hospitals used when $p_1 = p_2 = 1$, $g = 0$, $r_{i,1}$ = random int(1,50), $d_{i,1} = r_{i,1} + 5$, α_i = random int(1,20) , l_i= 2.

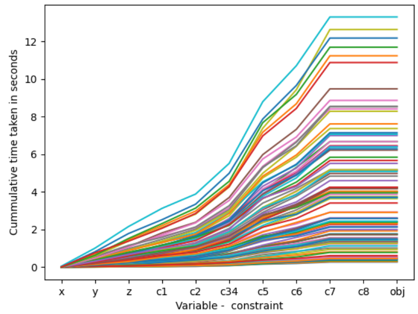

Figure 6 Time taken after computing each variable/constraint.

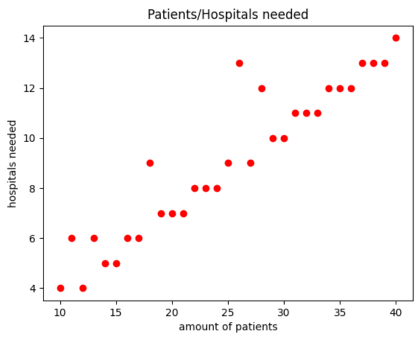

Figure 7 Hospitals used when $p_1 = 1$, $p_2 = 2$, $g = 0$, $r_{i,1}$ = random int(1,50), $d_{i,1}$ $= r_{i,1} + 5$, α_i = random int(1,20), $l_i = 2$.

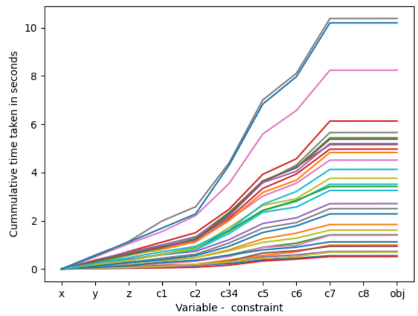

Figure 8 Time taken after computing each variable/constraint.

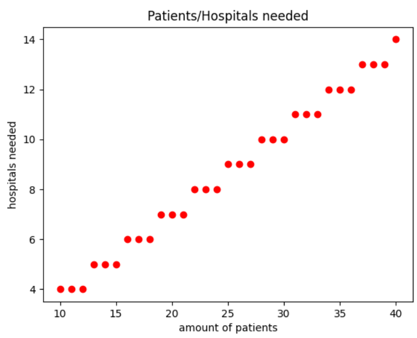

Figure 9 Hospitals needed when $p_1 = 1$, $p_2 = 2$, $g = 6$, $r_{i,1}$ = random int(1,50) , $d_{i,1}$ $= r_{i,1} + 5$, $\alpha_i = 0$, $l_i = 2$.

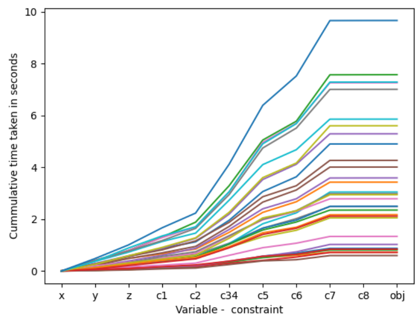

Figure 10 Time taken after computing each variable/constraint.

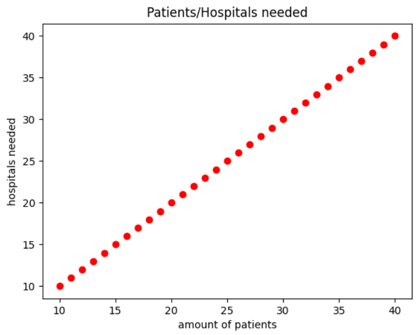

Figure 11 Hospitals used when $p_1 = 1$, $p_2 = 5$, $g = 0$, $r_{i,1}$ = random int(1,50), $d_{i,1} = r_{i,1}$, α_i = random int(1,20) , $l_i = 5$.

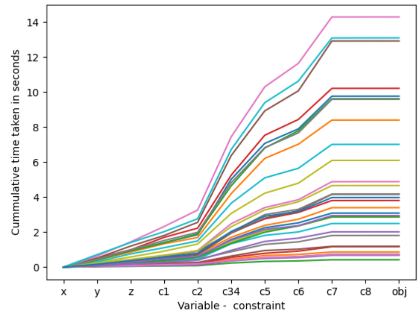

Figure 12 Time taken after computing each variable/constraint.

Explanation of results

- Figure 5 shows the results when considering the values: $p_1 = p_2 = 1$, $g = 0$, $r_{i,1}$ = random int(1,50), $d_{i,1} = r_{i,1} + 5$, α_i = random int(1,20) , l_i= 2. In this situation, patients have freedom of choosing when they want to receive the first dose in a range of 50 time slots as well as their desired delay until they get the second dose in a range of 20 time slots. We impose that $p_1 = p_2 = 1$ and $g = 0$ to simplify the experiment and we left a bit of room for the doses to be allocated inside the feasible intervals $|I_{i,1}| = 5$ and $|I_{i,2}| = l_i = 2$. This way the algorithm will have a more freedom of allocating patient doses inside of their correspondent feasible intervals. As we expected, (see Figure 5) the number of hospitals increases as the number of patients increase. But since the size of feasible intervals is larger than the processing times of the doses, the algorithm is able to exploit it and allocate patients in a reasonably small amount of hospitals. Again, an expected result. Figure 6 shows the cumulative time taken by the program to reach the solution. We can observe that the running time is quite fast (a little over 12' in the worst case).

- Figure 7 shows the results when considering the values: $p_1 = 1$, $p_2 = 2$, $g = 0$, $r_{i,1}$ = random int(1,50), $d_{i,1} = r_{i,1} + 5$, α_i = random int(1,20), $l_i = 2$. In this situation, patients have freedom of choosing when they want to receive the first dose in a range of 50 time slots as well as their desired delay until they get the second dose in a range of 20 time slots as in the previous case. We impose that $p_1 = 1$, $p_2 = 2$ and $g = 0$. In this case we forced the processing time of the second dose to be equal to the length of the second feasible interval for each patient, i.e. $|I_{i,2}| = l_i = 2$. This way the algorithm will have less freedom of allocating patient doses inside of their correspondent feasible intervals. As we expected, (see Figure 7) the number of hospitals increases as the number of patients increase faster than in the previous case. Since the size of the second feasible interval equals the processing of the second dose, the algorithm has to deal with a very restrictive situation and therefore needs more hospitals to allocate all patients. Again, an expected result. Figure 8 shows the cumulative time taken by the program to reach the solution. We can observe that the running time is quite fast (a little over 10' in the worst case).

- Figure 9 shows the results when considering the values: $p_1 = 1$, $p_2 = 2$, $g = 6$, $r_{i,1}$ = random int(1,50) , $d_{i,1} = r_{i,1} + 5$, $\alpha_i = 0$, $l_i = 2$. In this situation, patients have freedom of choosing when they want to receive the first dose in a range of 50 time slots but they don't have a choice on delaying the second dose more than the mandatory gap g which we have set to be 6 time slots. We impose that $p_1 = 1$ and $p_2 = 2$. In this case, as in the previous one we forced the processing time of the second dose to be equal to the length of the second feasible interval for each patient, i.e. $|I_{i,2}| = l_i = 2$. The algorithm will have less freedom of allocating patient doses inside of their correspondent feasible intervals but the two doses will be evenly spaced out. As we expected, (see Figure 9) the number of hospitals increases as the number of patients increase like in the previous case. It is really important to note that since the patient dependant delay is the same for every patient ($\alpha_i = 0$ and $g = 6$), the algorithm is able to exploit this fact and allocate patients in less hospitals when we compare it against the previous case. Figure 10 shows the cumulative time taken by the program to reach the solution. We can observe that the running time is quite fast (a little under 10' in the worst case).

- Figure 11 shows the results when considering the values: $p_1 = 1$, $p_2 = 5$, $g = 0$, $r_{i,1} =$ random int(1,50), $d_{i,1} = r_{i,1}$, $\alpha_i =$ random int(1,20) , $l_i = 5$. In this situation, patients have freedom of choosing when they want to receive the first dose in a range of 50 time slots as well as their desired delay until they get the second dose in a range of 20 time slots. We impose that $p_1 = 1$, $p_2 = 5$ and $g = 0$. In this case, we forced the processing time of the first and the second dose to be equal to the length of the first and the second feasible interval respectively. This way the algorithm will not have any freedom when allocating patient doses inside of their correspondent feasible intervals. Because $l_i = 5$ is big compared range of values in which patients can decide where to get vaccinated we expect the algorithm in this situation to exhibit a linear or almost linear relation between the patients and number of hospitals. As we expected, (see Figure 11) the number of hospitals increases as the number of patients increase linearly. Imposing such rigid restrictions force the optimal solution to take 1 hospital per patient. Figure 12 shows the cumulative time taken by the program to reach the solution. We can observe that the running time is quite fast (a little over 14' in the worst case).

We can conclude from this series of experiments that the offline algorithm proposed exhibits the expected behaviour in a wide range of situations.

5.3 Online Algorithm Results

This section we explore the performance of our proposed online algorithm. In the first place, we present a benchmark comparison between the two implementations of the online algorithm as discussed in section 4.2. In the second place, we present a comparison between the offline and online algorithm on a number of offline instances that where adapted to the online setting when run in the online algorithm. Last, we discuss how the two implementations behave on some of the online-specific instances.

5.3.1 Online benchmark results

The online implementations were timed on solving 50 randomly generated instances. Individual instances were generated using Python's builtin `random` class. Parameters were uniformly randomly chosen in the range of $[1, 100]$ for p_1, p_2, g. For patient i the availability has been uniformly randomly generated in the following range: $r_i = [1, 100]$, $d_i = r_i + p_1 + [1, 100]$, $\alpha_i = [0, 100]$, and $\ell_i = p_2 + [1, 100]$.

Algorithm 1 performs relatively well in terms of execution time, while algorithm 2 yields schedules requiring fewer hospitals. As can be seen in figure 13. A major issue for algorithm 2 is the second dose interval is dependent on when the first dose is scheduled. In its current

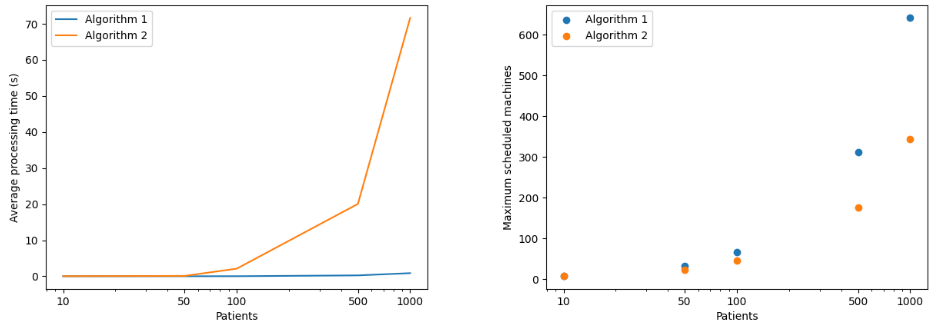

Figure 13 Average processing time (left) and maximum hospitals (right) across 50 random instances.

implementation algorithm 2 has to collect all free intervals for the first dose interval of patient i: $[r_i, d_i]$. Then, due to only committing to planning once the algorithm has found the optimal time slots for *both* doses, the possible time slots in which the second dose can be planned lies in the range $[r_i + p_1 + g + \alpha_i, d_i + p_1 + g + \alpha_i + \ell_i]$. This leads to a larger set of feasible time slots that need to be evaluated compared to algorithm 1. Combined with the nested iteration of intervals for both first and second dose this leads to comparatively bigger performance hits when encountering patients featuring more flexible schedules.

5.3.2 Comparison to offline results

Both online algorithms seem to be able to schedule the instances consistently in under a second on the tested offline instances. Table 1 shows a select set of offline instances tested, and the performance of both online algorithms on the correspondent adaptations. The number of hospitals the ILP solution uses is included for comparison. With the ILP solution as reference, algorithm 2 is able schedule patients optimally on a few of the instances. On instances with more patients the amount of hospitals required to schedule these patients tends to diverge from the number found by the ILP, Algorithm 1 often requires more hospitals than algorithm 2, but may perform better on certain instances that are punishing to the execution times of algorithm 2. Examples of such instances are 5-5, which caused algorithm 2 to time out by taking more than 120 seconds, yet algorithm 1 was able to solve the instance in 0.2 seconds.

5.3.3 Larger online instances

Figure 14 shows the amount of hospitals required to fulfill the schedules. The difference in the number of hospitals required per instance between the two algorithms seems to have no correlation with the size of the input. Algorithm 2 seems to again consistently require fewer hospitals to schedule an instance. Differences in computation times are negligible on the instances smaller than 5000 patients, all finish their computations in under 3 seconds, as can be seen in 15. On the 5000 patients instance algorithm 2 takes more than 120 seconds and is timed out. A possible reason for time out might be that due to the relatively large amount of patients and hospitals required, a lot of intervals across different hospitals will intersect with a patient's available times lots for the first dose. The algorithm calculates the maximum interval for the second dose, as explained in section 5.3.1 which in turn adds more potential

Instance	ILP	ALG 1		ALG 2	
	#M	#M	T(s)	#M	T(s)
3-1	2	**2**	0.0065	**2**	0.0060
4-3	TIME	TIME	>120	TIME	>120
4-4	2	3	0.0057	**2**	0.0060
5-3	3	4	0.0058	**3**	0.0060
5-5	TIME	2	0.2040	TIME	>120
6-1	2	4	0.0060	**2**	0.0061
7-2	1	3	0.0067	3	0.0064
10-1	3	5	0.0063	4	0.0065
45	5	11	0.7440	10	10.3232

Table 1 Hospital allocation comparison, on a select number of instances

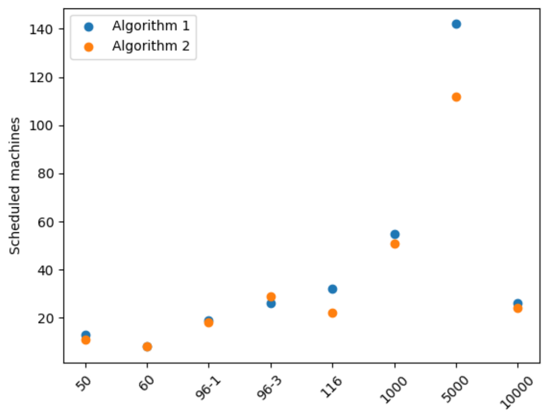

Figure 14 Hospitals scheduled per instance.

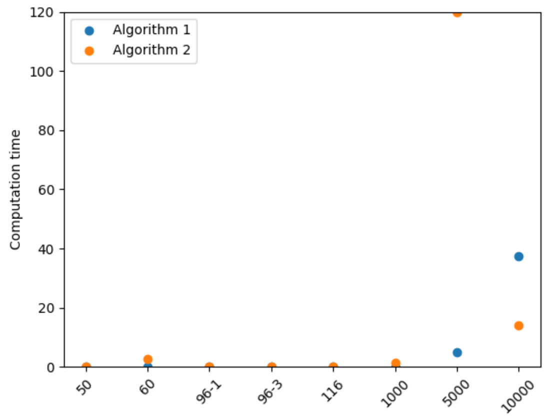

Figure 15 Computation time per instance.

intervals to be considered. Iterating through all these intervals to simultaneously find the optimal placement of two doses may then explain the increased processing times. On the 10000 patients instance algorithm 2 appears to have a faster running time than algorithm 1, which is contrary to previously observed behaviour. Due to the very small availability length for the second dose the additional computation time of finding an optimum for the first dose and largest range of the second dose may be smaller than the overhead the two individual interval collections and dose plannings that happen in algorithm 1.

5.3.4 Remarks on implementation

The two implementations of the online algorithm demonstrated show promising results, they manage to produce feasible schedules within reasonable time. Especially algorithm 1 performs very well if scheduling on an optimal amount of machines is not a priority. However, both algorithms suffer from several inefficiencies. The biggest inefficiency is the use of iteration to search for the optimal time slot in an interval. If these intervals become large this has a significant impact on performance, especially for algorithm 2. Instead a specialised function (e.g. a modified binary search) could reduce the amount of instructions 'wasted' on traversing each interval. Another limitation of the current implementation is that schedules currently feature a machine horizon beyond which no jobs can be scheduled. Due to this implementation it may not possible to schedule certain instances. Although it is technically possible to start using an infinite machine horizon this will require a substantial rewriting of the algorithms in the form of edge cases that need to be dealt with. Real world scenarios in which a scheduling program would benefit from an infinite machine horizon would be limited. Many publications on machine scheduling choose to work with a planning horizon, beyond which no schedule is planned. Therefore we argue that accounting for the possibility of infinitely long schedules is of little interest to us academically.

6 Conclusions

In this project we investigated the vaccine scheduling problem in the offline and online settings. With the research we conducted we have been able to provide exact solutions for small offline instances through integer linear programming and created a heuristic algorithm to provide solutions for online instances. For future research, we aim at improving the offline algorithm in order to be able to deal with larger instances as well as providing an upper bound for the competitive ratio of our proposed online algorithm. With respect to the offline algorithm, we encountered several difficulties when dealing with more than 50 patients. With respect to the online algorithm, we were able to provide a lower bound of $\lg(n)$ to the competitive ratio. Subjects for future work could include improving the running times of the online algorithms as discussed in section 5.3.4.

References

1 Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Seffi Naor. Machine minimization for scheduling jobs with interval constraints. In *45th annual IEEE symposium on foundations of computer science*, pages 81–90. IEEE, 2004. doi:10.1109/FOCS.2004.38.

2 Nikhil Devanur, Konstantin Makarychev, Debmalya Panigrahi, and Grigory Yaroslavtsev. Online algorithms for machine minimization. *arXiv preprint arXiv:1403.0486*, 2014.

3 G. Diepen, J. M. Van Den Akker, J. A. Hoogeveen, and J. W. Smeltink. Finding a robust assignment of flights to gates at amsterdam airport schiphol. *Journal of Scheduling*, 15(6):703–715, 2012. URL: www.scopus.com, doi:10.1007/s10951-012-0292-y.

4 Michael R. Garey, Ronald L. Graham, David S. Johnson, and Andrew Chi-Chih Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.

7 Appendix

Listing 1 Online Algorithm 1

```
plan_patient(S, r1, d1, x, ell)
    fs := get_free_schedule(S, (r1, d1));
    t1 := plan_dose(fs);
    S  := remove_planned_timeslot(S, t1, p1);
    fs := get_free_schedule(S, (t1+p1+x, t1+p1+x+ell));
    t2 := plan_dose(fs);
    S  := remove_planned_timeslot(S, t2, p2);
    return S, t1, t2;

plan_dose(fs)
    best_i:=-1;
    best_flexibility:=-1;
    foreach interval in fs do
    begin
        lower, upper:=interval;
        for i:=lower to upper do
        begin
            flexibility:=f(lower, i) + f(i+p1, upper);
            if(flexibility > best_flexibility):
                best_flexibility:=flexibility;
                best_i:=i;
            else:
                continue;
        end;
    end;
    return best_i;
```

Listing 2 Online Algorithm 2

```
plan_patient(S, r1, d1, x, ell)
    fs := get_free_schedule(S, (r1, d1), x, ell);
    t1 := plan_dose(fs, x, ell);
    S  := remove_planned_timeslots(S, p1);
    return S, t1, t2;

plan_dose(fs)
    best_i:=-1;
    best_j:=-1
    best_flexibility:=-1;
    foreach interval in fs do
    begin
        lower, upper:=interval;
        for i:=lower to upper do
        begin
            foreach interval in fs do
            lower2, upper2:=interval
            begin
                for j:=lower2 to upper2 do
                begin
                    flexibility:=f(lower, i) + f(i+p1, upper)
                               + f(lower2, j) + f(j+p2, upper2);
                    if(flexibility > best_flexibility):
                        best_flexibility:=flexibility;
                        best_i:=i;
                        best_j:=j;
                    else:
                        continue;
                end;
            end;
        end;
    end;
    return best_i, best_j;
```