Energy-efficient Automated Vertical Farms

This preprint is available at https://santini.in/

Energy-efficient Automated Vertical Farms

Maxence Delorme[1] and Alberto Santini[2]

[1]Department of Econometrics & Operations Research, Tilburg University, the Netherlands
[2]Department of Economics & Business, Universitat Pompeu Fabra, Spain; Data Science Centre, BGSE, Spain; ESSEC Business School, France; Institute for Advanced Studies, Paris Cergy Université, France

January 3, 2022

Abstract

Autonomous vertical farms (VFs) are becoming increasingly more popular because they allow to grow food minimising water consumption and the use of pesticides, while greatly increasing the yield per square metre compared with traditional agriculture. To meet sustainability goals, however, VFs must operate at maximum efficiency; it would be otherwise impossible to compete with the energy source powering plant growth in traditional agriculture: the sun. We introduce the Vertical Farming Elevator Energy Minimisation Problem (VFEEMP), which arises when minimising the energy consumption of automatic elevators servicing VFs. We prove that the decision problem associated with the VFEEMP is $\mathcal{NP}$-complete. To solve the problem, we propose three Mixed-Integer Linear Programming (MIP) formulations together with valid inequalities, and a Constraint Programming model. We present a large set of instances, both synthetic and derived from real-life data, and we determine through extensive computational experiments which instance characteristics have an impact on the difficulty of the problem and which formulations are the most suitable to solve the VFEEMP.

Keywords: vertical farming, task scheduling, operational research applications, integer linear programming, constraint programming.

1 Introduction

Access to high-quality food produced sustainably is a pressing problem in both developed and developing countries. Standard agricultural practices are not well-suited to scale for a world with a larger and increasingly urbanised population, less available water, and climate change [8]. Simply increasing the amount of land dedicated to agriculture is not a compelling long-term option because of the associated side-effects: deforestation, soil depletion, the need to use pesticides and fertilisers, emissions due to transport between the place of production and that of consumption, etc. Moreover, standard agriculture provides low yields in terms of the amount of nutrients produced per square meter of land used [4]. To mitigate the effects that the forecast increase of food demand will have on urban and rural communities, significant efforts are devoted to devise alternative agricultural practices. Among the main objectives pursued is to increase the yield per square meter and reduce externalities on the environment, while staying economically competitive [3].

Vertical Farming (VF), an agricultural technique which is gaining increasing traction, is establishing itself as an invaluable tool to meet the above objectives. VF consists of growing crops in stacked layers hosted in indoor support structures. The hosting infrastructure provides the plants with everything they need to grow in optimal conditions: the right amount of light and water, ventilation and appropriate CO_2 levels, nutrients, protection from pests, controlled temperatures. Such infrastructures vary considerably in terms of complexity, level of control they offer over the growth environment, and size [13, 12].

All types of structures, irrespective of their size or level of sophistication, provide some basic advantages compared to traditional farming: they protect the plants from weather and climate variability; they shelter them from parasites and pathogen; they allow to control the water and soil to ensure it is not affected by the presence of heavy metals or other dangerous substances; they

can be employed everywhere, thus bringing production closer to consumption and reducing transportation costs and emissions. For example, supermarkets, restaurants and hotels are increasingly adopting VF cabinets, approximately as large as standard refrigerators, to grow in-house herbs, leafy vegetables and berries. However, for VF to make a considerable impact towards achieving a more sustainable food supply chain and to be economically viable, operators strive to operate at larger scales [4]. Structures of the size of silos or re-purposed city buildings are more appropriate to host crops central to human diets, such as staple foods, vegetables and fruits [7, 2]. Although the technology which would allow such very large-scale projects to operate is not yet fully developed, interesting concepts are starting to emerge. For example, 6-to-12-metre tall "growth towers" in which plants grow under completely controlled conditions and with minimal human interaction [9] are available on the market. In such towers, each tray hosting a crop receives light, water, nutrients and ventilation from a computer-controlled Internet-of-Things system. Trays are moved around, irrigated, and inspected via automatic elevators equipped with water hoses and cameras. A human operator is needed to bring a new tray to the elevator's loading bay, and to collect it at the end of its growth period.

These large-scale systems allow a lower energetic footprint. For example, they are large enough to be equipped with solar panels for electricity production and contain enough air that water can be harvested from condensed humidity. In other respects, however, VF is still more energy-consuming than traditional farming. First, it uses artificial light to provide crops with an optimal amount of lighting and faster growth times. Although new technology such as LED lighting consumes significantly less energy compared to systems from just 5 years ago, it is clearly not as efficient as using sunlight. Second, for large-scale systems such as the growth towers described above, the automated elevators which serve the tower use a significant portion of the energy required by the structure.

The objective of this paper is to develop mathematical models and decision support tools to lower the energy consumption of the automatic elevators servicing large-scale VF structures. To do so, in the rest of this section, we give a precise description of the problem. Section 3 presents existing literature on the optimisation of vertical farms and on other related problems. We formalise our problem and prove its $\mathcal{NP}$-completeness in Section 4. In Section 5 we present three MIP formulations and a number of valid inequalities aimed at strengthening their linear relaxations, and we also present a formulation based on Constraint Programming (CP). After describing in Section 6 the instance sets we use, we present the results of a large set of computational experiments in Section 7. We conclude and point out future research directions in Section 8.

The main contributions of this paper are the following:

- We introduce a problem arising from the operation of autonomous vertical farms, which has timely applications in a rapidly growing industry.
- We prove that the problem is $\mathcal{NP}$-complete by reduction from the well-known SubsetSum problem.
- We present three MIP formulations and strengthen them with valid inequalities and variable fixing techniques, as well as a CP formulation.
- We generate and publish a large set of instances, both synthetic and derived from real-life data. We also determine which instance characteristics have an impact on the difficulty of the problem.
- We identify the best-performing formulation and set of valid inequalities. We also make public the source code of all our solvers.

2 Problem description

A VF tower is composed of stacked shelves in which a planner places trays containing crops. Each tray will spend some time in this system until the crops it contains are ready to be harvested. Each shelf can host one tray at a time and, after a tray leaves a shelf, another one can take its place. Trays become available from a "depot" at the bottom, spend the required amount of time on a shelf and, at the end of the crop's growth period, return to the depot. In an automatic vertical farm, an elevator moves the trays between the depot and the shelves. Figure 1 gives a schematic representation of a stack with five shelves (plus the depot) served by an elevator. A human operator brings a seeded tray to the depot so that the elevator can place it into a shelf. At the end of the growth cycle of the crop, the elevator brings the tray down and the operator picks it up from the depot and moves it into an area where the crop will be harvested.

The elevator also serves the trays performing activities such as watering and monitoring. For example, it can take pictures that a computer vision algorithm processes to ensure crops are growing

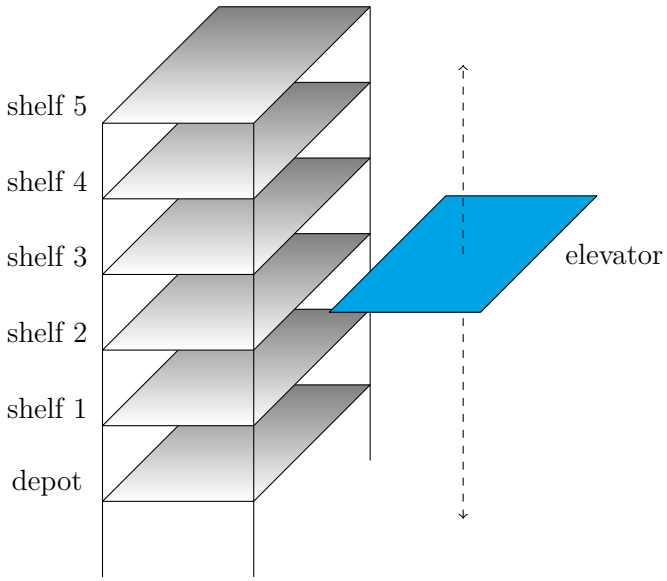

Figure 1: Schematic representation of a shelf tower with five shelves plus the depot, serve by an elevator.

well and are not affected by pests. Each tray has an associated list of tasks, and each task requires that the elevator travels to the corresponding shelf and spends some time there.

Each tray has a start time window during which it must be picked up at the depot to be placed on its assigned shelf. We define the time at which the elevator picks up the tray to bring it to its shelf as the tray's *start time*. There are also time windows related to each task and to the final pick up of the tray, but these are relative to the tray's start time. For example, if a tray has a start time window of one day (from hour 0 to 24), it can be brought to its shelf at hour 12. Then, if it must be watered after 6–8 hours from the growth start, this would translate into a task time window spanning from hour 18 to hour 20.

A computer system is in charge of deciding in which order the tasks should be completed, and correspondingly moving the elevator to perform each task. The objective of the system is to minimise the energy consumption of the elevator, making sure that all tasks are performed within their time windows. The elevator uses energy to travel between shelves. Therefore, the energy minimisation objective translates into the minimisation of the distance travelled by the elevator.

In the problem considered in this paper, another subsystem in the control software of the tower has already taken care of assigning each tray to the shelf it will occupy (see Section 8 for possible extensions). The elevator must then transport each tray to and from its assigned shelf, and visit the trays to perform the given list of tasks. The elevator can carry out tasks associated with different trays in any order, as long as the time windows are respected. On the other hand, tasks associated with the same tray must be carried out in the given order.

The objective of the problem is to determine a sequence of operations which minimises the travel distance of the elevator. Because the travel time of the elevator (a few seconds) is negligible compared to the time it must spend at the shelves to perform the tasks (several minutes), we can disregard it when timing the operations.

3 Literature review

There is a limited but growing amount of literature on the topic of the optimisation of VF. Two of the earliest works, by Yang, Hari, and Kuo [20] and Bennell, Martinez, and Potts [5], respectively, are only available as short abstracts in conference proceedings.

Yang, Hari, and Kuo [20] study the problem of selecting and scheduling a mix of crops to grow in a VF system at a tactical level. The objective is to maximise profits, taking into account that the price of each crop fluctuates during the year. The fluctuation is due to the possibility of growing some of the crops using open-air agriculture, which is highly seasonal. Therefore, it is more profitable to harvest a crop in the VF system during the period of the year when it is not available using open-air agriculture. The author proposes a small case study with a 2-year time horizon and 13 possible crops, which is solved using a MIP.

The problem of meeting as much demand as possible using crops grown in a VF cabinet was

studied by Bennell, Martinez, and Potts [5]. The authors consider a stack of shelves similar to the one presented in Figure 1, although smaller in scale: their proposed use-case is a cabinet placed directly at the consumer site (e.g., at a restaurant or a supermarket). In their settings, the capacity of each shelf and the growth speed of each crop are determined by a set of growth conditions, which the operator can change. For example, each crop can grow on a different growth medium, receive more or less light, water, etc. Using these factors to influence the length of the growth cycles, Bennell, Martinez, and Potts minimise the amount of unmet demand, weighted by crop prices. The authors propose a heuristic rolling-horizon algorithm.

Yang, Huang, and Ang [21] expand on [20], considering again the problem of selecting and scheduling crop growth in a VF system. The authors consider a setting in which crops receive sunlight (as opposed to artificial light) and the amount of light depends on the shelf's position and on whether the system operator installs rooftop solar panels, which obstruct the glass ceiling. Similar to [20], the authors aim at maximising the profit considering the seasonal variability of crop prices. Additionally, they introduce crop adjacency considerations, which make it more desirable for crops of the same family to grow on adjacent shelves. The MIP formulation proposed by the authors is not able to scale to realistically sized instances and, therefore, they implement a greedy heuristic which provides solutions for time horizons of up to 50 months.

Recently, Santini et al. [17] considered the problem of planning the production of crops on VF cabinets. The authors consider a system in which crops grow under controlled conditions (soil, water, light, CO_2 levels, etc.), which can change day-by-day and on a shelf-by-shelf level. They propose four objective functions which a planner might want to minimise: the amount of unmet demand (if it is not possible to satisfy the entire demand with the available infrastructure), the number of tray movements (trays are allowed to move from a shelf to another one during their growth), the number of shelf reconfigurations (i.e., the number of times the operator changes growth conditions for a shelf), and the number of shelves used. They prove that the problem is $\mathcal{NP}$-complete and provide MIP models for the four versions. The models are tested on instances with up to 6 crops, 12 shelves, and a time horizon of 100 days. Their results show that the choice of the objective function heavily influences both the computational performance of the models and the characteristics of the solution, hinting that a multi-objective approach could be useful to build a real-world decision support system.

In another recent paper, Cetegen and Stuber [6] present a robust optimisation model to design a VF system and schedule crop growth. The authors aim at building a portfolio of crops to grow, to minimise the risk of economic loss. Because controlled-environment agricultural techniques such as VF virtually eliminate risks on the grower's side (bad weather, pests, etc.) the authors consider risks coming from the market's side, i.e., demand and price variability. Using a cutting-plane method to solve a semi-infinite programme with semi-infinite constraints, the authors apply their algorithm to two case studies, validating the economic viability of using VF to grow crops such as lettuce, spinach, tomatoes, strawberries and mushrooms.

We also highlight a recent Master's thesis [1], which deals with task scheduling in a semi-automatic VF plant factory. The objective considered is the minimisation of the makespan of tasks to be performed by sensors and actuators on the crop trays growing in the VF system. This objective allows the authors to model the problem as a Flexible Job-Shop Scheduling Problem and use a variant of the Genetic Algorithm by Pezzella, Morganti, and Ciaschetti [14] to solve it.

In the broader context of reducing the environmental footprint of industrial production processes, we place our work within a growing number of papers which explicitly consider sustainability and energy efficiency when optimising production planning. For example, Wu and Che [19] propose a bi-objective flow-shop scheduling problem in which the two objectives are the minimisation of the total make-span, and the minimisation of energy consumption. Qiu, Qiao, and Pardalos [15] present a production routing problem in which excessive carbon emissions are penalised in the objective function, while reduced emissions yield a prize.

Finally, we remark that our problem has a superficial similarity with the Travelling Salesman Problem with Time Windows (TSPTW): if the task time windows were absolute rather than relative to the planting time, our problem could be seen as an extension of a 1-dimensional TSPTW. The current literature, however, does not present any work neither on routing problems in which the time window of a customer is relative to the visit time of another customer, nor on 1-dimensional special cases of routing problems with time windows.

4 Problem formalisation

Consider a tower with n shelves, making up set $S = \{1, \dots, n\}$. We denote the bottom depot as 0 and the set of all shelves plus the depot as $S' = \{0\} \cup S$. During the planning horizon, m trays will be placed on the shelves; we denote them as $T = \{1, \dots, m\}$. The time horizon itself is discretised in time intervals and denoted as $\Omega = \{1, \dots, \omega\}$.

Each tray $t \in T$ has a start time window $\left[\underline{w}_t^0, \bar{w}_t^0\right]$ (with $\underline{w}_t^0, \bar{w}_t^0 \in \Omega$ and $\underline{w}_t^0 \leq \bar{w}_t^0$) during which it must be picked up at the depot. The elevator must stay at the depot for $d_t^0 \geq 1$ additional time units when picking up the tray, to allow loading operations. The start time window is absolute. Conversely, the other time windows introduced below will be relative to the time in which the elevator reached the depot to pick up the tray. Furthermore, there is a list of l_t tasks which the elevator must perform on tray t; the set of these tasks is denoted as $J_t = \{1, \dots, l_t\}$. Each task $j \in J_t$ has an associated (relative) time window $\left[\underline{w}_t^j, \bar{w}_t^j\right]$ during which the elevator must start the task, and a duration $d_t^j \geq 1$ during which the elevator cannot move away from the shelf containing t. Finally, we denote with $\left[\underline{w}_t^{l_t+1}, \bar{w}_t^{l_t+1}\right]$ the (relative) time window during which the elevator must pick up tray t and bring it back to the depot, and with $d_t^{l_1+1} \geq 1$ the additional time the elevator must spend at the depot after bringing the tray, to allow unloading operations. We define the set of extended tasks as $J'_t = \{0\} \cup J_t \cup \{l_t + 1\}$.

We allow time windows to overlap for a given tray, i.e., it could happen that $\bar{w}_t^j + d_t^j \geq \underline{w}_t^{j+1}$, $\forall j \in \{1, \dots, l_t\}$. However, to ensure the proper growth of the crops, we require tasks to be executed in their given order, i.e., for a given tray, a task $j + 1$ cannot be performed before task j.

It is also possible that the allowed "shelf life" of a tray (i.e., the moments in which it is allowed to be present on its assigned shelf) can overlap with that of another tray assigned to the same shelf. In other words, given two trays t_1, t_2 such that both are assigned to the same shelf, it can happen that

$$\bar{w}_{t_1}^0 + \bar{w}_{t_1}^{l_{t_1}+1} \geq \underline{w}_{t_2}^0. \tag{1}$$

However, the two intervals $[\bar{w}_{t_1}^0, \underline{w}_{t_1}^0 + \underline{w}_{t_1}^{l_{t_1}+1}]$ and $[\bar{w}_{t_2}^0, \underline{w}_{t_2}^0 + \underline{w}_{t_2}^{l_{t_2}+1}]$ (if non-empty) cannot overlap, as they represent the time instants in which the trays are required to be on the shelf and two trays cannot occupy the same shelf at the same time.

Because all shelves are equally spaced in the silo, the distance travelled by the elevator when moving from a shelf $s_1 \in S'$ to another shelf $s_2 \in S'$ can be easily expressed as $c_{s_1,s_2} = c_{s_2,s_1} = |s_1 - s_2|$. We also denote the shelf assigned to a tray $t \in T$ as $s(t) \in S$.

We define the Vertical Farming Elevator Energy Minimisation Problem (VFEEMP) as the problem of deciding in which order to perform the tasks, to minimise the distance travelled by the elevator. Differently from any other problem we are aware of, most time windows are *relative* and depend on other decisions taken by the planner, namely the start time of the planting task. Because of this distinguishing feature, the VFEEMP seems to be neither a special case nor a generalisation of other known optimisation problems and, thus, a first interesting question regards its computational complexity. Theorem 1 answers this question, proving that the decision version of the VFEEMP is $\mathcal{NP}$-complete.

Theorem 1. *Let* VFEEMP-Decision *be the decision problem which asks whether there is a solution to the VFEEMP of cost at most x, for a given $x \in \mathbb{R}_0^+$. Then* VFEEMP-Decision *is $\mathcal{NP}$-complete.*

Proof. We prove the theorem by proving a stronger statement: that determining whether VFEEMP is feasible is itself $\mathcal{NP}$-complete. This clearly implies that VFEEMP-Decision is $\mathcal{NP}$-complete.

Consider an instance of the $\mathcal{NP}$-complete SubsetSum problem: given a set of n positive integers $\mathcal{B} = \{b_1, \dots, b_n\}$ and a positive integer c, the problem asks to determine whether there is a subset $\mathcal{S} \subseteq \mathcal{B}$ such that $\sum_{b \in \mathcal{S}} b = c$. We show how to create an instance of the VFEEMP such that, if the instance is feasible, then SubsetSum has answer Yes and, otherwise, it has answer No.

The main idea is to first transform the SubsetSum instance by multiplying each b_i and c by 2, to only deal with even numbers. Next, we build an instance in which each tray uses $2b_i$ units of elevator time, in which the elevator can never be idle, and in which there is a special tray which must be picked up at time $2c + 1$. In this way, being able to serve a subset of trays before time $2c + 1$ corresponds to finding a subset of the (doubled) integers summing up to $2c$, i.e., a set of the original integers which sums to c.

Formally, we create the VFEEMP instance as follows: let $B = 2\sum_{i=1}^{n} b_i$, set the number of shelves to 1, the number of trays to $n + 1$ and the time horizon to $\Omega = \{1, \dots, B + 4\}$. Each of the first $i = 1, \dots, n$ trays has:

- Start time window $\left[1, B - 2b_i\right]$ and $d_i^0 = b_i$.

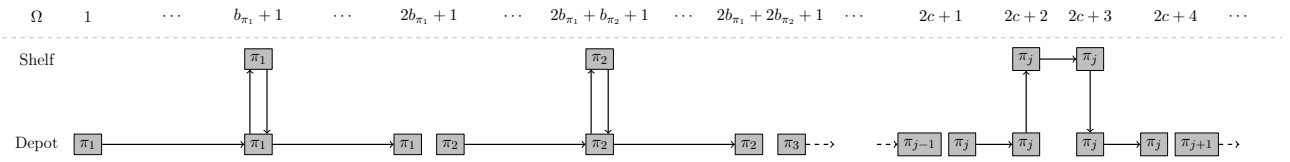

Figure 2: Visualisation of a feasible VFEEMP transformed from SubsetSum.

- No intermediate tasks.
- End time window $[b_i, b_i]$ and $d_i^1 = b_i$, meaning that the end task must be processed right after the start task..

Tray $n+1$ has:

- Start time window $[2c+1, 2c+1]$ and $d_{n+1}^0 = 1$.
- A unique task of duration $d_{n+1}^1 = 1$ with time window $[1, 1]$. In our instance this relative time window is equivalent to absolute time window $[2c+2, 2c+2]$.
- End time window $[1, 1]$ and $d_{n+1}^2 = 1$. In our instance this relative time window is equivalent to absolute time window $[2c+3, 2c+3]$.

Because of the characteristics of the instance, finding a solution amounts to determining a permutation of the indices $\{1, \ldots, n+1\}$ indicating in which order the trays occupy the only shelf. The upper bounds on the start time windows of the shelves imply that the elevator cannot be idle at any time in a feasible solution, or there would be no time to serve all trays: each tray $i \neq n+1$ requires $2b_i$ units of time and tray $n+1$ requires 3 time intervals so that the elevator must be busy up to the last time interval $B+4$.

If VFEEMP is feasible, let $(\pi_1, \ldots, \pi_{n+1})$ be the permutation corresponding to a solution and let j be the index such that $\pi_j = n+1$. Then, because each tray i consumes exactly $2b_i$ units of elevator time and because tray $n+1$ must be picked up exactly at time $2c+1$, then we must have that $\sum_{i=1}^{j-1} 2b_i = 2c$. This implies $\sum_{i=1}^{j-1} b_i = c$ and, therefore, SubsetSum has the answer Yes. Figure 2 depicts such a situation.

Noting that the above VFEEMP instance can be built in linear time in the size n of the SubsetSum instance concludes the proof. □

5 Model formulation

In this section, we propose four formulations for the VFEEMP. The first three are based on MIP: the first uses a polynomial number of variables and disjunctive constraints; the second one uses a pseudo-polynomial number of variables; the third one is a hybrid which only uses binary variables from the first and the second formulation. Finally, the fourth formulation is based on CP.

5.1 A model with disjunctive constraints

To model the fact that the elevator starts and must return to the depot, we first add to Ω a dummy time interval 0, and to T a dummy tray with index 0, assigned to the depot with start time window $[0, 0]$ and $d_0^0 = 1$, no intermediary tasks, and end time window $[\omega, \omega]$ and $d_0^1 = 1$.

We introduce variables $y_{t_1,j_1,t_2,j_2} \in \{0, 1\}$, for $t_1, t_2 \in T$, $j_1 \in J'_{t_1}$ and $j_2 \in J'_{t_2}$ (with $j_1 \neq j_2$ when $t_1 = t_2$). Variable y_{t_1,j_1,t_2,j_2} takes value 1 iff the elevator performs task j_1 on tray t_1 and, immediately afterwards, it performs task j_2 on tray t_2. Recall that the assignment of trays to shelves is given, and that we denote with $s(t) \in S$ the shelf assigned to tray $t \in T$. We denote with $c(y_{t_1,j_1,t_2,j_2}) \in \mathbb{N}$ the travel cost associated with variable y_{t_1,j_1,t_2,j_2}. To assign a value to $c(\cdot)$, we consider the following cases:

$$
\begin{array}{lll}
j_1 = 0 & j_2 = 0 & c(y_{t_1,0,t_2,0}) = s(t_1) + s(t_2) \quad (2)\\
j_1 = 0 & j_2 \in J_{t_2} & c(y_{t_1,0,t_2,j_2}) = |s(t_1) - s(t_2)| \quad (3)\\
j_1 = 0 & j_2 = l_{t_2} + 1 & c(y_{t_1,0,t_2,l_{t_2}+1}) = |s(t_1) - s(t_2)| + s(t_2) \quad (4)\\
j_1 \in J_{t_1} & j_2 = 0 & c(y_{t_1,j_1,t_2,0}) = s(t_1) + s(t_2) \quad (5)\\
j_1 \in J_{t_1} & j_2 \in J_{t_2} & c(y_{t_1,j_1,t_2,j_2}) = |s(t_1) - s(t_2)| \quad (6)\\
j_1 \in J_{t_1} & j_2 = l_{t_1} + 1 & c(y_{t_1,y_1,t_1,l_{t_2}+1}) = |s(t_1) - s(t_2)| + s(t_2) \quad (7)\\
j_1 = l_{t_1} + 1 & j_2 = 0 & c(y_{t_1,l_{t_1}+1,t_2,0}) = s(t_2) \quad (8)
\end{array}
$$

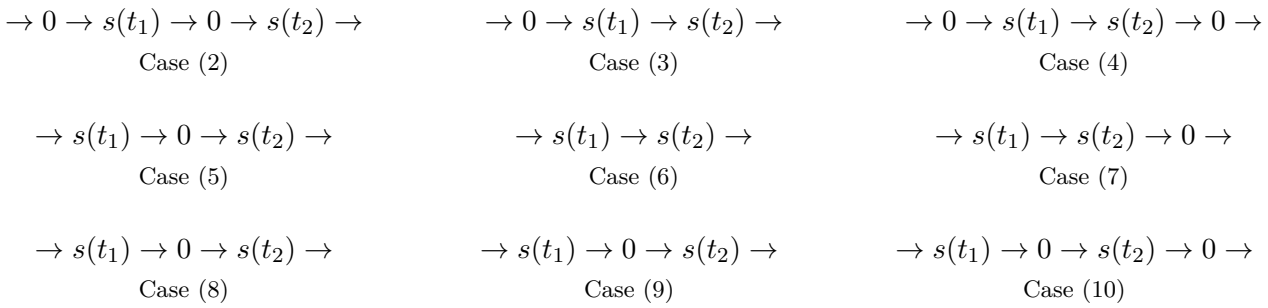

Figure 3: Possible cases for the costs associated with variables y_{t_1,j_1,t_2,j_2}.

$$j_1 = l_{t_1} + 1 \quad j_2 \in J_2 \qquad c(y_{t_1,l_{t_1}+1,t_2,j_2}) = s(t_2) \tag{9}$$

$$j_1 = l_{t_1} + 1 \quad j_2 = l_{t_2} + 1 \quad c(y_{t_1,l_{t_1}+1,t_2,l_{t_2}+1}) = 2s(t_2). \tag{10}$$

The elevator movements associated with each of these cases are shown in Figure 3. Note that a case in the figure depicts more movements than those considered in the corresponding cost to prevent counting some movement costs twice. For example, in Case (10) we do not count the cost of bringing back the elevator to the depot after picking up tray t_1 because it is already counted in the previous move, which was either a Case (4), a Case (7), or another Case (10).

We also introduce the following notation:

$$\alpha_{tj} = \begin{cases} \underline{w}_t^0 & \text{if } j = 0 \\ \underline{w}_t^0 + \underline{w}_t^j & \text{otherwise,} \end{cases} \tag{11}$$

$$\beta_{tj} = \begin{cases} \bar{w}_t^0 & \text{if } j = 0 \\ \bar{w}_t^j + \bar{w}_t^0 & \text{otherwise.} \end{cases} \tag{12}$$

In other words, α_{tj} and β_{tj} are the earliest and latest time intervals at which task $j \in J'_t$ can start. We now notice that not all possible quadruples (t_1, j_1, t_2, j_2) are valid indices for variables y:

- If $t_2 = 0$ and $j_2 = 0$, then no indices t_1, j_1, t_2, j_2 can be valid, as the start task of the dummy tray cannot have any predecessor.
- Analogously, if $t_1 = 0$ and $j_1 = 1$, then no indices t_1, j_1, t_2, j_2 can be valid, because the end task of the dummy tray cannot have any successor.
- We can also omit indices which would result in time window violations. Specifically, if $\beta_{t_2 j_2} < \alpha_{t_1 j_1} + d_{t_1}^{j_1}$ then task j_2 must be performed before task j_1 and cannot be its successor. Therefore, we need not consider indices t_1, j_1, t_2, j_2 satisfying the above condition.
- If $t_1 = t_2$, then only indices for which $j_2 = j_1 + 1$ are valid; otherwise, tasks would be performed in the wrong order for that tray.
- If there is another task $j_3 \in J'_{t_3}$ (for some tray $t_3 \in T$) which must take place between j_1 and j_2, then these two tasks cannot happen in succession. More precisely, indices t_1, j_1, t_2, j_2 are not valid if

$$\exists t_3 \in T,\ j_3 \in J'_{t_3} \ :\ \alpha_{t_3 j_3} + d_{t_3}^{j_3} > \beta_{t_1 j_1} \text{ and } \beta_{t_3 j_3} < \alpha_{t_2 j_2} + d_{t_2}^{j_2}.$$

Let us denote with V the set of valid indices for variables y, i.e., the set of all quadruples (t_1, j_1, t_2, j_2) which are not excluded by one of the above conditions.

We also consider continuous variables s_{tj} indicating the starting time of task j on tray t ($t \in T, j \in J'_t$). Finally, let $\Delta_{t_1 t_2} \in \{0, 1\}$ be a binary variable defined for each pair of trays $t_1, t_2 \in T$ such that $s(t_1) = s(t_2)$. It will take value 1 iff tray t_1 is placed on the shelf before tray t_2. The VFEEMP can then be modelled with the following MIP, which we denote **M1**:

$$\min \quad \sum_{(t_1,j_1,t_2,j_2) \in V} c(y_{t_1,j_1,t_2,j_2}) \cdot y_{t_1,j_1,t_2,j_2} \tag{13}$$

$$\text{s.t.} \quad \sum_{(t_1,j_1,t,j) \in V} y_{t_1,j_1,t,j} = 1 \qquad \forall t \in T,\ \forall j \in J'_t, (t,j) \neq (0,0) \tag{14}$$

$$\sum_{(0,0,t,j) \in V} y_{0,0,t,j} = 1 \tag{15}$$

$$\sum_{(t_1,j_1,t,j)\in V} y_{t_1,j_1,t,j} = \sum_{(t,j,t_2,j_2)\in V} y_{t,j,t_2,j_2} \quad \forall t \in T \setminus \{0\},\ \forall j \in J'_t \tag{16}$$

$$s_{t0} \geq \underline{w}_t^0 \quad \forall t \in T \tag{17}$$

$$s_{t0} \leq \bar{w}_0^j \quad \forall t \in T \tag{18}$$

$$s_{tj} \geq s_{t0} + \underline{w}_t^j \quad \forall t \in T,\ \forall j \in J'_t \setminus \{0\} \tag{19}$$

$$s_{tj} \leq s_{t0} + \bar{w}_t^j \quad \forall t \in T,\ \forall j \in J'_t \setminus \{0\} \tag{20}$$

$$s_{tj} \geq s_{t,j-1} + d_t^{j-1} \quad \forall t \in T,\ \forall j \in J'_t \setminus \{0\} \tag{21}$$

$$s_{t_2j_2} \geq s_{t_1j_1} + d_{t_1}^{j_1} - M(1 - y_{t_1,j_1,t_2,j_2}) \quad \forall (t_1,j_1,t_2,j_2) \in V \tag{22}$$

$$s_{t_10} \geq s_{t_2,l_{t_2}+1} + d_{t_2}^{l_{t_2}+1} - M \cdot \Delta_{t_1t_2} \quad \forall t_1,t_2 \in T,\ s(t_1) = s(t_2),\ t_1 \neq t_2 \tag{23}$$

$$\Delta_{t_1t_2} + \Delta_{t_2t_1} \leq 1 \quad \forall t_1,t_2 \in T,\ s(t_1) = s(t_2),\ t_1 < t_2 \tag{24}$$

$$y_{t_1,j_1,t_2,j_2} \in \{0,1\} \quad \forall (t_1,j_1,t_2,j_2) \in V \tag{25}$$

$$s_{tj} \geq 0 \quad \forall t \in T,\ \forall j \in J'_t \tag{26}$$

$$\Delta_{t_1t_2} \in \{0,1\} \quad \forall t_1,t_2 \in T\ :\ s(t_1) = s(t_2),\ t_1 \neq t_2 \tag{27}$$

The objective function (13) minimises the travel cost of the elevator. Constraints (14) ensure that every task of every tray (excluding the start task of the dummy tray) has a predecessor and, therefore, is processed. Constraints (15) force the elevator to process the start task of the dummy tray, and constraints (16) make sure that every task (excepted those associated with the dummy tray) has a direct predecessor and a direct successor. Constraints (17) and (18) make sure that each tray is picked up during its associated (absolute) start time window, and constraints (19) and (20) ensure that every task other than the start task takes place during its associated (relative) time window. Constraints (21) force tasks associated with the same tray to be scheduled in the given order and are necessary because task time windows can overlap. Constraints (22), in which M is a sufficiently large number, force the starting time of any task to be not earlier than the starting time plus the duration of its predecessor. The tightest value for M is $\beta_{t1j1} + d_{t_1}^{j_1} - \alpha_{t_2j_2}$. Note that constraints (22) could easily be updated to incorporate a travelling time between shelves $s(t_1)$ and $s(t_2)$ if the time spent in elevator movements was not negligible compared to the time spent performing the tasks. Constraints similar to (22) are also used in the literature on routing problems with time windows to sequence customer visits (see, e.g., constraint (14) of [11]) and in scheduling to sequence tasks (see, e.g., constraint (5) of [18]). Constraints (23) link variables s and Δ (again, using a sufficiently large number M), and constraints (24) ensure that no two trays occupy the same shelf at the same time. The tightest value for M in (23) is $\beta_{t_2,l_{t_2}+1} + d_{t_2}^{l_{t_2}+1} - \alpha_{t_10}$. Variables Δ and constraints (23)–(24) are required because tray shelf lives can overlap and, therefore, the model must determine a processing order for trays on the same shelf. Finally, (25), (26), and (27) are variable domain definitions.

For inequalities (22) and (23) we use big-M constraints over SOS/indicator constraints because, experimentally, the M factors do not cause numerical difficulties. Indeed, we use constraint-specific M values resulting in coefficients that are not much larger than the other coefficients in the model.

5.1.1 Valid inequalities for M1

We introduce the following sets of valid inequalities for model **M1**. Their impact is tested by means of computational experiments presented in Section 7.1.

Task incompatibility (TaskInc). The task incompatibility considerations we use to reduce the set of feasible indices for variables y can be extended to more than two tasks at a time. For example, the following inequality is valid for model **M1**:

$$y_{t_1,j_1,t_2,j_2} + y_{t_2,j_2,t_3,j_3} \leq 1 \quad \forall (t_1,j_1,t_2,j_2),(t_2,j_2,t_3,j_3) \in V\ :\ \beta_{t_3j_3} < \alpha_{t_1j_1} + d_{t_1}^{j_1} + d_{t_2}^{j_2}. \tag{28}$$

In this case, (28) states that tasks j_1, j_2 and j_3 cannot be performed in the given order, due to their time windows and durations. Although one could consider similar clique inequalities for larger sets of mutually incompatible tasks, detecting such sets becomes harder while the constraints would be less likely to be active. For this reason, we only add triangle inequalities of type (28) to model **M1**.

Force task sequence (TaskSeq). While constraint (28) *excludes* incompatible tasks, we next present a constraint aimed at *forcing* tasks to happen one immediately after the other. In other

words, we look for suitable conditions on $(t_1, j_1, t_2, j_2) \in V$ which force $y_{t_1,j_1,t_2,j_2} = 1$. These conditions must ensure that (i) j_2 cannot start before j_1, and that (ii) no other task j_3 (relative to some tray t_3) can start between j_1 and j_2. A sufficient condition for (i) is that starting j_2 at its earliest possible time would make the elevator miss j_1's time window. A sufficient condition for (ii) is that there is no valid start time for j_3 which is compatible with both j_1's and j_2's time windows. According to these considerations, we introduce the following valid variable fixing equality for model **M1**:

$$\begin{aligned} y_{t_1,j_1,t_2,j_2} = 1 \quad & \forall (t_1, j_1, t_2, j_2) \in V \text{ such that:} \\ & \alpha_{t_2 j_2} + d_{t_2}^{j_2} > \beta_{t_1 j_1} \text{ and} \\ & \forall t_3 \in T,\ \forall j_3 \in J'_{t_3},\ (t_3, j_3) \neq (t_1, j_1),\ (t_3, j_3) \neq (t_2, j_2) \\ & \nexists k \in \left[\alpha_{t_3 j_3}, \beta_{t_3 j_3}\right] \ :\ k \geq \alpha_{t_1 j_1} + d_{t_1}^{j_1} \text{ and } k + d_{t_3}^{j_3} \leq \beta_{t_2 j_2}. \end{aligned} \tag{29}$$

Furthermore, if condition (ii) holds but condition (i) does not, then no task j_3 can start between tasks j_1 and j_2; however any of j_1 and j_2 can precede the other. In this case, if condition (ii) also holds when swapping j_1 and j_2 (that is, if no task j_3 can start between j_2 and j_1, as well), then we can add the following constraint:

$$\begin{aligned} y_{t_1,j_1,t_2,j_2} + y_{t_2,j_2,t_1,j_1} = 1 \quad & \forall (t_1, j_1, t_2, j_2) \in V \text{ such that} \\ & \forall t_3 \in T,\ \forall j_3 \in J'_{t_3},\ (t_3, j_3) \neq (t_1, j_1),\ (t_3, j_3) \neq (t_2, j_2) \\ & \nexists k \in \left[\alpha_{t_3 j_3}, \beta_{t_3 j_3}\right] \ :\ k \geq \alpha_{t_1 j_1} + d_{t_1}^{j_1} \text{ and } k + d_{t_3}^{j_3} \leq \beta_{t_2 j_2} \text{ and} \\ & \nexists k \in \left[\alpha_{t_3 j_3}, \beta_{t_3 j_3}\right] \ :\ k \geq \alpha_{t_2 j_2} + d_{t_2}^{j_2} \text{ and } k + d_{t_3}^{j_3} \leq \beta_{t_1 j_1}. \end{aligned} \tag{30}$$

Constraint (30) states that either j_1 is performed immediately before j_2, or j_2 is performed immediately after j_1. It thus excludes the possibility that the elevator performs the third task between j_1 and j_2, no matter in which order j_1 and j_2 are performed.

2- and 3-cycle elimination (CycElim). We next propose two families of valid inequalities similar to cycle elimination constraints in routing and elementary shortest path problems [see, e.g., 10]:

$$y_{t_1 j_1 t_2 j_2} + y_{t_2 j_2 t_1 j_1} \leq 1 \quad \forall (t_1, j_1, t_2, j_2) \in V \ :\ (t_2, j_2, t_1, j_1) \in V \tag{31}$$

$$y_{t_1 j_1 t_2 j_2} + y_{t_2 j_2 t_3 j_3} + y_{t_3 j_3 t_1 j_1} \leq 2 \quad \forall (t_1, j_1, t_2, j_2), (t_2, j_2, t_3, j_3) \in V \ :\ (t_3, j_3, t_1, j_1) \in V \tag{32}$$

Constraint (31), a 2-cycle elimination constraint, states that either j_1 takes place before j_2, or j_2 takes place before j_1. Constraint (32), a 3-cycle elimination constraint, extends the same reasoning to three tasks at a time. It forbids solutions inducing a "cycle" with j_1 performed before j_2, j_2 before j_3, and j_3 before j_1.

Bounds on task start times (STBound). The next family of valid inequalities aims at bounding variables s_{tj} both from above and below. Given a task $j \in J'_t$, we first consider the set Π_{tj} of tray-tasks pairs containing tasks which the elevator cannot perform *after* j and must, therefore, precede j: $\Pi_{tj} = \{(t', j') \ :\ \alpha_{tj} + d_t^j > \beta_{t'j'} \in V\}$. For any task $j' \in J'_{t'}$, time instant $\alpha_{t'j'} + d_{t'}^{j'}$ denotes the earliest possible finish time of task j'. Taking the maximum such value among tasks $j' \in \Pi_{tj}$ ensures that the elevator cannot be free to perform task j before this time. Therefore, the following inequality is valid:

$$s_{tj} \geq \max_{(t', j') \in \Pi_{tj}} \left\{\alpha_{t'j'} + d_{t'}^{j'}\right\} \quad \forall t \in T,\ \forall j \in J'_t. \tag{33}$$

Analogously, we can consider the set of tray-task pairs which the elevator cannot perform *before* j and must, therefore, follow j: $\Sigma_{tj} = \{(t', j') \ :\ \alpha_{t'j'} + d_{t'}^{j'} > \beta_{tj} \in V\}$. Then, the following inequality, analogous to (33), is valid:

$$s_{tj} + d_t^j \leq \min_{(t', j') \in \Sigma_{tj}} \left\{\beta_{t'j'}\right\} \quad \forall t \in T,\ \forall j \in J'_t. \tag{34}$$

Minimum inter-task time (MinIT). If a task $j' \in J_{t'}$ must follow another task $j \in J_t$ (i.e., if $(j', t') \in \Sigma_{jt}$) then, j' cannot start before $s_{jt} + d_t^j$. Therefore, the following constraint is valid:

$$s_{t'j'} \geq s_{tj} + d_t^j \quad \forall t \in T,\ \forall j \in J_t,\ \forall (t', j') \in \Sigma_{jt}. \tag{35}$$

Note that inequality (35) is not implied by (22) and (33). Furthermore, in the continuous relaxation of the problem, fractional values of y can produce a solution in which (34) is satisfied, but (35) is violated.

5.2 A model with a pseudo-polynomial number of variables

We introduce parameter $s(t,j)$ for $t \in T$ and $j \in J'_t$, which represents the shelf where the elevator is located after performing task j:

$$s(t,j) = \begin{cases} s(t) & \text{if } j \neq l_t + 1 \\ 0 & \text{if } j = l_t + 1 \end{cases}.$$

We now define a binary variable x_{tjk} for $t \in T$, $j \in J'_t$ and $\alpha_{tj} \leq k \leq \beta_{tj}$. Variable x_{tjk} takes value 1 iff the elevator starts performing task j on tray t at time k. We also define an integer variable z_k ($k \in \Omega \setminus \{1\}$) indicating the cost of the movements made between time $k-1$ and time k. Finally, we introduce a binary variable p_{ik} ($i \in S'$ and $k \in \Omega$) taking value 1 iff the elevator is at shelf i at time k.

Then model **M2** reads as follows:

$$\min \quad \sum_{k=2}^{\omega} z_k \tag{36}$$

$$\text{s.t.} \quad \sum_{k=\alpha_{tj}}^{\beta_{tj}} x_{tjk} = 1 \qquad \forall t \in T,\ \forall j \in J'_t \tag{37}$$

$$\sum_{i=0}^{n} p_{ik} = 1 \qquad \forall k \in \Omega \tag{38}$$

$$\sum_{t \in T} \sum_{j \in J'_t} \sum_{k_1 \in \Omega_{tjk}} x_{tjk_1} \leq 1 \qquad \forall k \in \Omega \tag{39}$$

$$\sum_{t \in T} \sum_{\substack{j \in J'_t \\ \alpha_{tj} \leq k \leq \beta_{tj} \\ s(t,j)=i}} x_{tjk} \leq p_{ik} \qquad \forall i \in S',\ \forall k \in \Omega \tag{40}$$

$$x_{tjk} \leq \sum_{\substack{k_1 \in \left[\underline{w}_t^0, \bar{w}_t^0\right] \\ \underline{w}_t^j \leq k - k_1 \leq \bar{w}_t^j}} x_{t0k_1} \qquad \forall t \in T,\ \forall j \in J'_t \setminus \{0\},\ \forall k \in \left[\alpha_{tj}, \beta_{tj}\right] \tag{41}$$

$$x_{tjk} \leq \sum_{\substack{k_1 \in \left[\alpha_{t,j-1}, \beta_{t,j-1}\right] \\ d_t^{j-1} \leq k - k_1}} x_{t,j-1,k_1} \qquad \forall t \in T,\ \forall j \in J'_t \setminus \{0\},\ \forall k \in \left[\alpha_{tj}, \beta_{tj}\right] \tag{42}$$

$$x_{t0k} \leq 1 - \sum_{\substack{t_1 \in T \setminus \{t\} \\ s(t_1) = s(t)}} \left(\sum_{k_1 = \alpha_{t_1 0}}^{\min\{k, \beta_{t_1 0}\}} x_{t_1 0 k_1} - \sum_{k_1 = \alpha_{t_1, l_{t_1}+1}}^{\min\{k, \beta_{t_1, l_{t_1}+1}\}} x_{t_1, l_{t_1}+1, k_1} \right) \forall t \in T,\ \forall k \in \left[\alpha_{t0}, \beta_{t0}\right] \tag{43}$$

$$z_k \geq \sum_{i=0}^{n} i p_{i,k-1} - \sum_{i=0}^{n} i p_{ik} + \sum_{\substack{t \in T \\ \alpha_{t0} \leq k \leq \beta_{t0}}} 2 \cdot s(t) \cdot x_{t0k} \qquad \forall k \in \Omega \tag{44}$$

$$z_k \geq \sum_{i=0}^{n} i p_{ik} - \sum_{i=0}^{n} i p_{i,k-1} + \sum_{\substack{t \in T \\ \alpha_{t,l_t+1} \leq k \leq \beta_{t,l_t+1}}} 2 \cdot s(t) \cdot x_{t,l_t+1,k} \qquad \forall k \in \Omega \tag{45}$$

$$x_{tjk} \in \{0,1\} \qquad \forall t \in T,\ \forall j \in J'_t,\ \forall k \in \left[\alpha_{tj}, \beta_{tj}\right] \tag{46}$$

$$p_{ik} \in \{0,1\} \qquad \forall i \in S',\ \forall k \in \Omega \tag{47}$$

$$z_k \geq 0 \text{ and integer} \qquad \forall k \in \Omega, \tag{48}$$

where Ω_{tjk} contains the starting time indexes of task j of tray t for which the task would not be completed by time k. The objective function (36) minimises the travel cost of the elevator. Constraints (37) ensure that every task of every tray is processed (exactly once), and constraints (38) make sure that the elevator is at a unique position at any moment. Constraints (39) forbid the elevator to process more than one task at once. Constraints (40) force the elevator to be at the shelf assigned to the tray whose task it is starting to process. These constraints, together with (39) and objective function (36) ensure that the elevator will not move for the entire duration of the task. Constraints (41) ensure that each task (except the start task) is started during its relative time window. These constraints also implicitly ensure that the start task begins within its absolute time window. Constraints (42) force the tasks to be scheduled in the given order, and are required because the time windows of consecutive tasks can overlap. Constraints (43) make sure that the starting task of a tray can only take place once its associated shelf is free. We need these constraints because the allowed shelf lives can overlap for trays assigned to the same shelf. Finally, constraints (44) and (45) compute the cost of the moves performed at every time instant (for $k = 1$, we use the convention $p_{i,k-1} = 0$), while (46)–(48) are variable domain definitions.

5.2.1 Valid inequalities for M2

We adapt the families of valid inequalities STBound and MinIT devised for **M1** (see Section 5.1.1).

Valid inequalities (33) and (34) result in fixing x variables to 0 in **M2**:

$$x_{tjk} = 0 \qquad \forall t \in T,\ \forall j \in J'_t,\ \forall k \in \left[\alpha_{tj}, \beta_{tj}\right] \ :\ k < \max_{(t',j') \in \Pi_{tj}} \left\{\alpha_{t'j'} + d^{j'}_{t'}\right\} \tag{49}$$

$$x_{tjk} = 0 \qquad \forall t \in T,\ \forall j \in J'_t,\ \forall k \in \left[\alpha_{tj}, \beta_{tj}\right] \ :\ k > \min_{(t',j') \in \Sigma_{tj}} \left\{\beta_{t'j'}\right\} - d^{j}_{t}. \tag{50}$$

Inequality (35) corresponds to the following constraint:

$$x_{tjk} \leq \sum_{\substack{k_1 \in \left[\alpha_{t'j'}, \beta_{t'j'}\right] \\ d^{j'}_{t'} \leq k - k_1}} x_{t'j'k_1} \quad \forall t \in T,\ \forall j \in J'_t,\ \forall k \in \left[\alpha_{tj}, \beta_{tj}\right],\ \forall (t', j') \in \Pi_{jt}. \tag{51}$$

5.3 A hybrid model

We now introduce a model which uses both binary variables y_{t_1,j_1,t_2,j_2} of **M1** and x_{tjk} of **M2**, and does not need any other set of variables. We denote this model as **M3**.

$$\min \quad \sum_{(t_1,j_1,t_2,j_2) \in V} c(y_{t_1,j_1,t_2,j_2}) \cdot y_{t_1,j_1,t_2,j_2} \tag{52}$$

$$\text{s.t.} \quad \sum_{(t_1,j_1,t,j) \in V} y_{t_1,j_1,t,j} = 1 \qquad \forall t \in T, \forall j \in J'_t, (t,j) \neq (0,0) \tag{53}$$

$$\sum_{(0,0,t,j) \in V} y_{0,0,t,j} = 1 \tag{54}$$

$$\sum_{(t_1,j_1,t,j) \in V} y_{t_1,j_1,t,j} = \sum_{(t,j,t_2,j_2) \in V} y_{t,j,t_2,j_2} \qquad \forall t \in T \setminus \{0\}, \forall j \in J'_t \tag{55}$$

$$\sum_{k=\alpha_{tj}}^{\beta_{tj}} x_{tjk} = 1 \qquad \forall t \in T, \forall j \in J'_t \tag{56}$$

$$\sum_{t \in T} \sum_{j \in J'_t} \sum_{k_1 \in \Omega_{tjk}} x_{tjk_1} \leq 1 \qquad \forall k \in \Omega \tag{57}$$

$$x_{tjk} \leq \sum_{\substack{k_1 \in \left[\underline{w}^0_t, \bar{w}^0_t\right] \\ \underline{w}^j_t \leq k - k_1 \leq \bar{w}^j_t}} x_{t0k_1} \qquad \forall t \in T, \forall j \in J'_t \setminus \{0\}, \forall k \in \left[\alpha_{tj}, \beta_{tj}\right] \tag{58}$$

$$x_{tjk} \leq \sum_{\substack{k_1 \in \left[\alpha_{t,j-1},\beta_{t,j-1}\right] \\ d_t^{j-1} \leq k-k_1}} x_{t,j-1,k_1} \qquad \forall t \in T,\ \forall j \in J'_t \setminus \{0\},\ \forall k \in \left[\alpha_{tj},\beta_{tj}\right] \tag{59}$$

$$x_{t0k} \leq 1 - \sum_{\substack{t_1 \in T \setminus \{t\} \\ s(t_1)=s(t)}} \left(\sum_{k_1=\alpha_{t_1 0}}^{\min\{k,\beta_{t_1 0}\}} x_{t_1 0 k_1} - \sum_{k_1=\alpha_{t_1,l_{t_1}+1}}^{\min\{k,\beta_{t_1,l_{t_1}+1}\}} x_{t_1,l_{t_1}+1,k_1} \right) \forall t \in T, \forall k \in \left[\alpha_{t0},\beta_{t0}\right] \tag{60}$$

$$y_{t_1,j_1,t_2,j_2} + x_{t_1 j_1 k_1} \leq 1 \qquad \forall (t_1,j_1,t_2,j_2) \in V,\ \forall k_1 \in \Gamma^1_{t_1,j_1,t_2,j_2} \tag{61}$$

$$y_{t_1,j_1,t_2,j_2} + x_{t_1 j_1 k_1} + \sum_{\substack{k_2 \in \left[\alpha_{t_2 j_2},\beta_{t_2 j_2}\right] \\ k_2 < k_1 + d_{t_1}^{j_1}}} x_{t_2 j_2 k_2} \leq 2 \qquad \forall (t_1,j_1,t_2,j_2) \in V,\ \forall k_1 \in \Gamma^2_{t_1,j_1,t_2,j_2} \tag{62}$$

$$y_{t_1,j_1,t_2,j_2} \in \{0,1\} \qquad \forall (t_1,j_1,t_2,j_2) \in V, \tag{63}$$

$$x_{tjk} \in \{0,1\} \qquad \forall t \in T, \forall j \in J'_t, \forall k \in \left[\alpha_{tj},\beta_{tj}\right]. \tag{64}$$

In the formulation of **M3** we used the following sets:

$$\Gamma^1_{t_1,j_1,t_2,j_2} = \{1 + \beta_{t_2 j_2} - d_{t_1}^{j_1}, \ldots, \beta_{t_1 j_1}\},$$
$$\Gamma^2_{t_1,j_1,t_2,j_2} = \left\{\alpha_{t_1 j_1}, \ldots, \min\{\beta_{t_1 j_1}, \beta_{t_2 j_2} - d_{t_1}^{j_1}\}\right\}.$$

The objective function (52) and constraints (53)–(55) are the same as, respectively, objective (13) and constraints (14)–(16) of **M1**. Analogously, constraints (56)–(60) are the same as (37)–(43) of **M2**. The only new constraints are (61) and (62), which link variables x and y. In particular, (61) prevents the elevator from starting a task j_1 at time k_1 if it is the predecessor of another task j_2 that has to be started at time $k_1 + d_{t_1}^{j_1} - 1$ or before. Analogously, (62) forbids j_1 to start at time k_1 if it is the predecessor of another task j_2 starting at a time k_2 such that $k_1 + d_{t_1}^{j_1} > k_2$. Inequalities (28)–(32) and (49)–(51) are also valid for **M3**. Note that if the travelling time between shelves is not negligible, we only have to change (61) by adjusting the range of coefficient k_2 in the condition "$k_2 < k_1 + d_{t_1}^{j_1}$".

5.4 A constraint programming model

Our CP model, **M4**, uses two sets of variables. The first set is comprised of integer variables $s_{tj} \in \left[\alpha_{tj},\beta_{tj}\right]$ indicating the starting time of task $j \in J'_t$ of tray $t \in T$. The second set is made of interval sequence variables $A(t,j)$, with $A(t_1,j_1) = (t_2,j_2)$ iff the elevator performs task j_2 on tray t_2 immediately after performing task j_1 on tray t_1. The model reads as follows:

$$\min \quad \sum_{t \in T} \sum_{j \in J'_t} c(y_{t,j,A(t,j)}) \tag{65}$$

$$\text{s.t.} \quad s_{tj} \geq s_{t0} + \underline{w}_t^j \qquad \forall t \in T,\ \forall j \in J'_t \setminus \{0\} \tag{66}$$

$$s_{tj} \leq s_{t0} + \bar{w}_t^j \qquad \forall t \in T,\ \forall j \in J'_t \setminus \{0\} \tag{67}$$

$$s_{tj} \geq s_{t,j-1} + d_t^{j-1} \qquad \forall t \in T,\ \forall j \in J'_t \setminus \{0\} \tag{68}$$

$$\text{NoOverlap}\left\{[s_{tj}, s_{tj} + d_t^j], \forall t \in T,\ \forall j \in J'_t\right\} \tag{69}$$

$$s_{t_1 0} \geq s_{t_2,l_{t_2}+1} + d_{t_2}^{l_{t_2}+1} \ \text{ OR } \ s_{t_2 0} \geq s_{t_1,l_{t_1}+1} + d_{t_1}^{l_{t_1}+1} \qquad \forall t_1,t_2 \in T,\ s(t_1) = s(t_2),\ t_1 \neq t_2 \tag{70}$$

$$s_{tj} \in \left[\alpha_{tj},\beta_{tj}\right] \qquad \forall t \in T, \forall j \in J'_t \tag{71}$$

$$A(t_1,j_1) \in \{(t_2,j_2) : t_2 \in T,\ j_2 \in J'_{t_2},\ (t_1,j_1) \neq (t_2,j_2)\} \forall t_1 \in T,\ \forall j_1 \in J'_{t_1}. \tag{72}$$

The objective function (65) minimises the travel cost of the elevator by using the interval sequence variables to compute the appropriate values of function $c(\cdot)$ introduced in Section 5.1.

Interval sequence variables also ensure that all but one task (the last of the sequence) have a direct successor and that every task is processed. Constraints (66) and (67) make sure that every task (except the start tasks) takes place during its associated (relative) time window. Constraints (68) force tasks associated to the same tray to be scheduled in the given order. Constraint (69) forbids the elevator to process more than one task at a given moment. Constraints (70) ensure that no two trays occupy the same shelf at the same time. Domain constraints (71) and (72) define the variable domains, with (71) also ensuring that every starting task is processed during its absolute time window.

Implementation details. We solve model **M4** with the `Cplex` constraint programming optimiser (see Section 7). We define each task as an `IloIntervalVar` variable and model the domain constraints with functions `setStartMin` and `setStartMax`. Precedence constraints between the starting task and the other tasks of the same tray are enforced using `IloStartBeforeStart` and `IloStartOf`: the former to forbid that task start too early and the latter to forbid that they start too late. We model precedence constraints between two consecutive tasks of the same tray using `IloEndBeforeStart`. We used `IloStartOf`, `IloEndOf`, and the logical operator "`||`" to model constraints (70). We embed the `IloIntervalVar` variables in an `IloIntervalSequenceVar` variable, which we use to compute the objective function, with the help of the `IloTypeOfNext` function. Finally, we model the non-overlap constraint using `IloNoOverlap`.

6 Instance generation

We generated two sets of instances, named Synthetic and Realistic. The Synthetic set provides a wide variety of instance parameters (time horizon length, number of shelves and trays, number and duration of tasks, etc.) in order to test the quality of the proposed formulations on a diverse test body. The Realistic set provides instances which closely resemble real-life applications in VF silos.

6.1 Synthetic Instances

We describe the process used to generate feasible synthetic instances. Note that, because the VFEEMP is $\mathcal{NP}$-complete, verifying the feasibility of an instance is, in general, hard. Therefore, we approach the instance generation problem starting from a backbone instance which is guaranteed to be feasible and transforming it into a complete instance while preserving feasibility. We use three phases to generate an instance:

- We first generate a sequence of tasks with their respective start times and durations as they would appear in a feasible solution, i.e., we guarantee that the elevator can perform all tasks. We call this structure the *backbone instance*.
- Next, we assign the tasks to the trays and the trays to shelves, ensuring that no two trays occupy the same shelf at the same time.
- Finally, we add time windows around the start times generated to allow for optimisation opportunities and reordering of tasks.

Generating task start times and durations. Let $m \in \mathbb{N}$ be the number of trays in the instance, $\omega \in \mathbb{N}$ be the time horizon length, and $\delta \in \mathbb{N}$ be the target number of tasks to assign to each tray. We first create a list of $\eta := m\delta$ tasks specifying, for each of them, their start time and their duration. Task start times are obtained by randomly and uniformly sampling η time instants from set $\{1, \dots \omega - 1\}$; we denote the start times as $k^{\mathrm{s}}_1, \dots, k^{\mathrm{s}}_\eta$. Without loss of generality, let $k^{\mathrm{s}}_j < k^{\mathrm{s}}_{j+1}$ for all $j \in \big[1, \eta - 1\big]$. Next, we assign each task a duration. We first sample a tentative duration d^{s}_j uniformly at random in $\big[1, \max\{1, \lfloor \omega/\eta \rfloor\}\big]$. If $d^{\mathrm{s}}_j > k^{\mathrm{s}}_{j+1}$, i.e., if two tasks would overlap, we set $d^{\mathrm{s}}_j = k^{\mathrm{s}}_{j+1}$.

Assigning tasks to trays. We first assign the start task of the trays; afterwards, we assign all other tasks. For each tray $t \in \big[1, m\big]$, we select a task uniformly at random as t's start task. In this phase we exclude from the possible tasks those whose start time is too close to the end of the time horizon; to this end, we do not consider tasks j such that $k^{\mathrm{s}}_j > \frac{4}{5}\omega$. We remove all tasks assigned as start tasks from the list of available tasks. Next, we assign $\delta - 1$ further tasks to each tray. Let $j^{\mathrm{s}}_t \in \{1, \dots, \eta\}$ be the index of the task chosen as start task for t and recall that tasks are indexed in increasing order of their start time. For each tray t, we scan available tasks one by

one starting from $j_t^{\mathrm{s}}+1$ (or whatever is the first available task after j_t^{s}). When scanning a task, we assign it to tray t with a probability $\pi^{\mathrm{gr}} \in (0,1)$. In case the task is assigned, we remove it from the list of assigned tasks; otherwise, we move on to scanning the next available task until we add all $\delta - 1$ remaining tasks to tray t. Probability π^{gr} measures how "greedily" we assign consecutive tasks to the same tray. A high value of π^{gr} gives trays with shorter and non-overlapping shelf lives; conversely, low values of π^{gr} result in trays with longer and overlapping shelf lives.

Assigning trays to shelves. We assign trays to shelves greedily. We first sort trays by their shelf life start and then assign each tray to the available shelf with the lowest index. A shelf is available if it is not occupied by another tray at that moment. The number of shelves is not bounded a priori, but we note that high values of π^{gr} allow more trays to occupy the same shelf (during different periods, of course) and result in a number of shelves typically between $\frac{m}{2}$ and $\frac{3m}{4}$ in our instances. Low values of π^{gr} reduce the probability that two trays can be assigned to the same shelf and result in a number of shelves typically between $\frac{3m}{4}$ and m.

Generating time windows. The last phase in our instance generation procedure consists of generating time windows around the task start times of the backbone instance. Given a task j, its time window is $\left[k_j^{\mathrm{s}} - \xi, k_j^{\mathrm{s}} + \xi\right]$ and time windows for non-start tasks are adjusted compared to the tray planting time to make them relative. The time window half-width is defined as $\xi := \gamma \frac{\omega}{m\delta}$, where $\gamma \geq 1$ is a parameter. Larger values of γ result in larger and more frequently overlapping time windows, which give more feasible task orderings for the elevator.

Parameters used. Table 1 describes the parameters used for instance generation. We generate an instance for each possible combination of the parameters, which vary independently. This gives a total of $3^5 = 243$ instances in the SYNTHETIC set.

Parameter	Description	Values
m	Number of trays	$14, 16, 18$
ω	Time horizon length	$150, 200, 250$
δ	Tasks per tray	$4, 5, 6$
π^{gr}	Greediness of task assignment	$0.2, 0.4, 0.6$
γ	Time window size multiplier	$1.0, 1.5, 2.0$

Table 1: Parameters used in the generation of the SYNTHETIC instance set.

Figure 4 shows the optimal solution of an example instance generated with parameters $m = 16$, $\omega = 250$, $\delta = 5$, $\pi^{\mathrm{gr}} = 0.6$, $\gamma = 1.5$. The x axis represents the time instants and the y axis the shelf number. Each yellow box denotes the "shelf life" of a tray, whose progressive id number is reported next to the top-left corner of the box. The lines represent the elevator moving and performing tasks (in black) or being idle (in red).

6.2 Realistic instances

To generate the REALISTIC set, we used real-life confidential data about the growth cycle of six crops and the relative tasks to be performed when they grow in a VF silo. Compared to the SYNTHETIC set, these instances have much larger time horizons, larger shelf lives and time windows, and more tasks per tray. Time horizons range from 15 to 128 days which, at a resolution of one time instant per 15 minutes, corresponds to values of ω between 1440 and 12 288. The instances contain either 5, 10, 15, or 20 shelves and between 5 and 40 trays. Each tray has, on average, 115.4 assigned tasks (including the start and end tasks).

The REALISTIC instances are particularly challenging because the number of variables and constraints in our models grows as a function of either the number of trays, the number of tasks, the time horizon length, the time windows sizes, or a combination of these factors. Furthermore, large time windows with significant overlaps contribute to the difficulty of these instances (see Section 7.1.3).

Figure 5 shows the optimal solution of a REALISTIC instance with six trays and a time horizon of 80 days. Comparing Figure 4 and Figure 5, we highlight the differences in the length of the time horizon, the average shelf life, and the number of tasks associated with each tray.

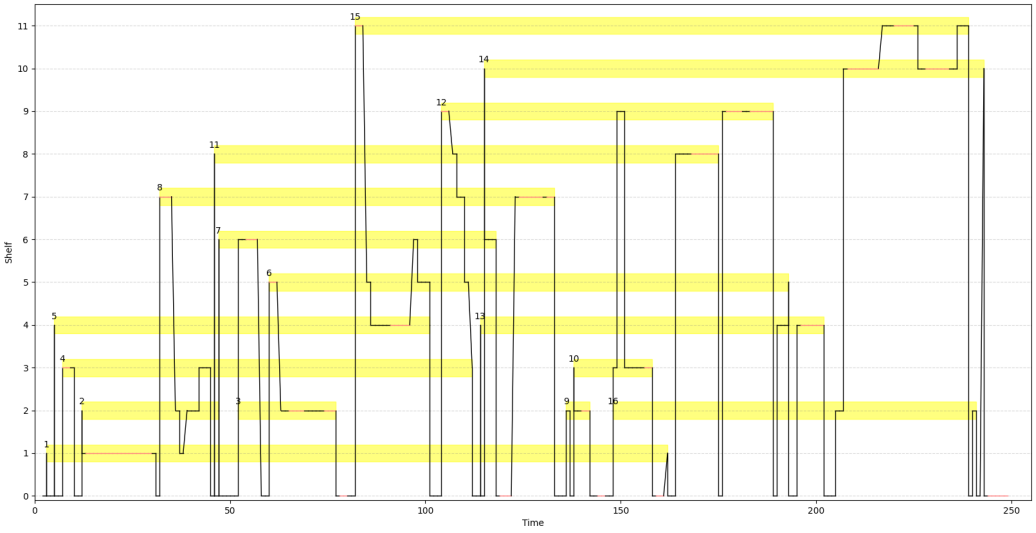

Figure 4: Optimal solution of a SYNTHETIC instance with parameters $m = 16$, $\omega = 250$, $\delta = 5$, $\pi^{\text{gr}} = 0.6$, $\gamma = 1.5$.

7 Computational results

In this section, we present insights on the behaviour of our models on both the SYNTHETIC and REALISTIC instances presented in Section 6. We analyse the impact of the valid inequalities introduced in Section 5 and identify instance characteristics which make the problem harder to solve. All models were implemented in `C++`; we solved the MIP models using `Gurobi` version 9 and the CP model using `Cplex` version 12.1. We make available on GitHub [16] the instance generator, the solver source code and the instance files. We performed the computational experiments on a 4-core machine equipped with an Intel Xeon CPU runing at 2.4GHz and 4GB RAM.

7.1 Results on the Synthetic instances

We used the synthetic instances as a test bed to investigate:

- The impact of valid inequalities TASKINC, TASKSEQ, CYCELIM, STBOUND, and MINIT on the linear relaxation of the three MIP formulations.
- The impact of the above valid inequalities on the branch-and-bound algorithm used by `Gurobi` to solve the integer models.
- How instance characteristics, determined by the instance generation parameters, affect the runtimes and optimality gaps of the MIP formulations.
- How the CP approach compares with the best MIP model.

7.1.1 Impact of valid inequalities

In Section 5, we introduced five families of valid inequalities. All five of them are applicable to models **M1** and **M3**, while only inequalities STBOUND and MINIT are valid for model **M2**.

We first measure the impact that each inequality, in isolation, has on the value of the continuous relaxation of the three models. To this end, we consider the value of the relaxation gap defined as $\text{Gap}\% = 100 \cdot (\text{UB} - \text{LB})/\text{UB}$ where, for each instance, UB represents the best-known primal solution and LB is the objective value of the continuous relaxation. We present these results in Tables 2 to 4, where columns "T" report the time in seconds needed to solve the continuous relaxation and columns "#" list the number of valid inequalities added. Some columns "#" are omitted in Table 4 because the values for TASKINC, TASKSEQ and CYCELIM are the same as in Table 2, while the values for STBOUND are the same as in Table 3. Values for MINIT are different between models **M2** and **M3** because the latter uses the dummy tray 0 while the former does not. Instances are grouped by two generation parameters, γ and δ, which have an important impact on the difficulty of the instances (see Section 7.1.3). Each row refers to the average over all instances sharing the

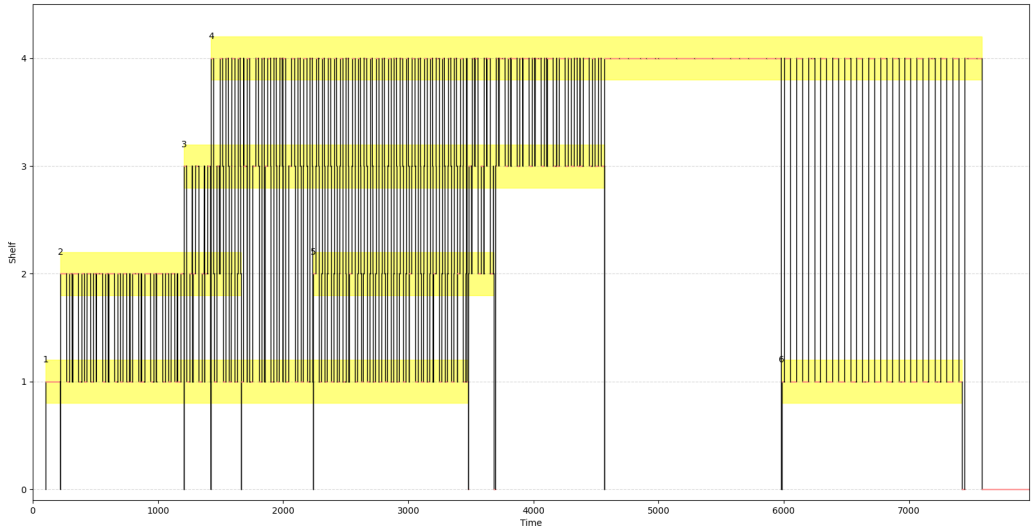

Figure 5: Optimal solution of a REALISTIC instance with six trays and a time horizon of 80 days (7680 time instants).

		NONE		TASKINC			TASKSEQ			CYCELIM			STBOUND			MINIT		
γ	δ	Gap%	T	Gap%	T	#	Gap%	T	#	Gap%	T	#	Gap%	T	#	Gap%	T	#
1.0	4	4.141	0.2	3.767	0.2	129	4.086	0.2	19	3.981	0.3	803	3.985	0.2	3	3.896	0.3	2009
	5	4.553	0.2	4.219	0.2	147	4.359	0.2	22	4.453	0.3	877	4.412	0.2	2	4.321	0.3	3097
	6	6.435	0.2	5.898	0.2	191	6.416	0.2	28	6.347	0.3	1071	6.404	0.3	3	6.345	0.4	4343
	All	5.025	0.2	4.612	0.2	156	4.935	0.2	23	4.909	0.3	917	4.915	0.2	3	4.835	0.3	3150
1.5	4	3.412	0.2	3.328	0.3	235	3.407	0.2	14	3.422	0.3	1544	3.432	0.2	2	3.430	0.3	1929
	5	5.421	0.2	5.015	0.3	311	5.270	0.3	20	5.309	0.4	1994	5.248	0.3	2	5.224	0.4	2997
	6	7.872	0.3	7.163	0.3	386	7.875	0.3	25	7.803	0.5	2367	7.918	0.3	1	7.849	0.6	4280
	All	5.568	0.2	5.169	0.3	311	5.517	0.3	20	5.511	0.4	1968	5.532	0.3	2	5.501	0.4	3069
2.0	4	4.220	0.3	4.059	0.3	379	4.277	0.3	14	4.335	0.4	2647	4.283	0.3	2	4.315	0.4	1885
	5	6.880	0.3	6.515	0.3	530	6.872	0.3	19	6.900	0.5	3439	6.807	0.3	2	6.865	0.5	2980
	6	8.170	0.4	7.840	0.4	643	8.158	0.4	26	8.320	0.6	4091	8.140	0.4	2	8.284	0.6	4211
	All	6.402	0.3	6.117	0.4	517	6.414	0.3	20	6.496	0.5	3392	6.388	0.3	2	6.466	0.5	3025
All		5.665	0.3	5.299	0.3	328	5.622	0.3	21	5.638	0.4	2093	5.612	0.3	2	5.600	0.4	3081
				-0.366	+0.0		-0.043	+0.0		-0.027	+0.1		-0.053	+0.0		-0.065	+0.2	

Table 2: Impact of valid inequalities on the strength of the continuous relaxation of model **M1**.

same parameter values for γ and δ. The last row reports, for each valid inequality, the percentage gap decrease and the root runtime variation, over all instances of the SYNTHETIC set.

The columns labelled with NONE, which report the results of the base formulations without valid inequalities, make clear that model **M1** has the strongest relaxation, followed by **M3**. Model **M2** exhibits large root gaps, which do not decrease adding valid inequalities.

The impact of valid inequalities on the other two models is positive. For **M1**, inequalities TASKINC have the strongest effect. Other inequalities only slightly decrease the root gaps. On the other hand, adding any type of inequality does not significantly increase the runtime at the root node. Note that we generate very few inequalities of family STBOUND; despite their number, they have a small but non-negligible impact on the root gap.

Valid inequalities have a larger effect on model **M3**, although not enough to make it competitive with **M1**. As for **M1**, inequalities TASKINC are effective; for model **M3**, however, CYCELIM are even more effective and achieve a root gap reduction of almost 0.6 percentage points. Inequalities MINIT slightly reduce the gaps but significantly increase the runtime, due to their high number.

		None		STBound			MinIT		
γ	δ	Gap%	T	Gap%	T	#	Gap%	T	#
1.0	4	16.937	1.0	16.937	1.1	4	16.937	3.6	14844
	5	18.450	1.1	18.450	1.1	3	18.450	4.6	19033
	6	23.015	1.2	23.015	1.2	3	23.015	6.1	24356
	All	19.423	1.1	19.423	1.1	3	19.423	4.7	19411
1.5	4	11.635	1.2	11.635	1.2	3	11.635	5.4	18641
	5	15.581	1.3	15.581	1.4	2	15.581	7.5	25383
	6	20.992	1.5	20.992	1.5	1	20.992	9.3	31710
	All	16.070	1.4	16.070	1.3	2	16.070	7.4	25245
2.0	4	10.487	1.6	10.487	1.6	2	10.487	8.7	23212
	5	13.854	1.9	13.854	1.9	2	13.842	12.2	31817
	6	17.320	1.8	17.320	1.9	2	17.320	14.5	38684
	All	13.844	1.8	13.844	1.8	2	13.840	11.8	31238
All		16.444	1.4	16.444	1.4	3	16.443	8.0	25298
				-0.000	+0.0		-0.001	+6.5	

Table 3: Impact of valid inequalities on the strength of the continuous relaxation of model **M2**.

		None		TaskInc		TaskSeq		CycElim		STBound		MinIT		
γ	δ	Gap%	T	Gap%	T	Gap%	T	Gap%	T	Gap%	T	Gap%	T	#
1.0	4	5.573	0.5	5.249	0.5	5.551	0.5	4.865	0.5	5.509	0.5	5.509	1.5	15373
	5	6.404	0.5	5.953	0.5	6.363	0.5	5.416	0.5	6.373	0.5	6.373	1.9	19602
	6	8.357	0.5	7.779	0.5	8.362	0.6	7.256	0.6	8.357	0.5	8.373	2.4	24969
	All	6.758	0.5	6.309	0.5	6.739	0.5	5.828	0.5	6.726	0.5	6.731	1.9	19981
1.5	4	4.721	0.6	4.288	0.7	4.721	0.7	4.314	0.7	4.721	0.7	4.721	2.7	19319
	5	6.773	0.8	6.294	0.8	6.750	0.7	6.116	0.8	6.773	0.8	6.773	3.5	26123
	6	9.888	0.9	9.331	0.8	9.888	0.8	9.314	1.0	9.888	0.8	9.888	4.1	32494
	All	7.127	0.8	6.638	0.8	7.120	0.7	6.581	0.8	7.127	0.7	7.127	3.4	25979
2.0	4	5.504	1.0	5.147	1.0	5.504	1.0	5.295	1.1	5.504	1.0	5.504	4.1	24053
	5	8.375	1.1	7.776	1.1	8.375	1.1	8.050	1.3	8.375	1.1	8.375	5.5	32737
	6	9.841	1.2	9.421	1.2	9.841	1.2	9.433	1.5	9.841	1.2	9.841	6.7	39648
	All	7.882	1.1	7.423	1.1	7.882	1.1	7.570	1.3	7.882	1.1	7.882	5.4	32146
All		7.255	0.8	6.789	0.8	7.246	0.8	6.659	0.9	7.245	0.8	7.247	3.6	26035
				-0.466	+0.0	-0.009	+0.0	-0.596	+0.1	-0.011	+0.0	-0.009	+2.8	

Table 4: Impact of valid inequalities on the strength of the continuous relaxation of model **M3**.

7.1.2 Branch-and-bound performance

A tighter continuous relaxation does not necessarily correspond to a MIP which is easier to solve. Therefore, we use the results obtained in Section 7.1.1 as an indication, but still perform further experiments to determine the impact of valid inequalities on the overall branch-and-bound algorithm used to solve the three formulations. Because evaluating all possible subsets of inequalities to activate would be too time consuming, we add them incrementally. For each model, we start from the one which reduced the average percentage gap the most according to the results presented in Section 7.1.1, and proceed in decreasing order.

For each model, we report results relative to two formulations. The first is the base model without valid inequalities, under header Base. The second is the formulation, among those we test following the above procedure, which gives the lowest average gap (see below for a precise definition of the gap). For model **M2** the base formulation also gives the lowest gap.

Table 5 reports the results of our experiments. Columns "Gap%" list the percentage optimality gap obtained with a time limit of one hour. The gap is defined as $100 \cdot (\text{UB} - \text{LB})/\text{UB}$, where UB and LB are, respectively, the best primal and dual bounds obtained within the time limit. Columns "T" report the runtime in seconds, while columns "N" list the number of branch-and-bound nodes visited. Columns "Opt%" give the percentage of instances solved to optimality.

For model **M1**, we obtain the best results when adding inequalities TaskInc and MinIT; for model **M3**, when adding inequalities TaskInc and CycElim.

Model **M1** clearly outperforms the other two formulations. When strengthened with the valid inequalities, in fact, **M1** solves almost 99% of the Synthetic instances to optimality, with an average runtime of 72.021 seconds. Model **M3** (with valid inequalities) also shows good performance,

		M1 Base				**M1** + TaskInc + MinIT				**M2** Base				**M3** Base				**M3** + TaskInc + CycElim			
γ	δ	Gap%	T	N	Opt%	Gap%	T	N	Opt%	Gap%	T	N	Opt%	Gap%	T	N	Opt%	Gap%	T	N	Opt%
1.0	4	0.000	2.305	1538	100.00	0.000	2.623	2011	100.00	0.087	3554.730	799868	3.70	0.000	30.289	5852	100.00	0.000	25.556	2074	100.00
	5	0.000	2.997	2440	100.00	0.000	2.937	1992	100.00	0.070	3404.087	739683	11.11	0.000	31.963	4875	100.00	0.000	21.819	2989	100.00
	6	0.000	6.590	5760	96.30	0.000	6.037	4068	96.30	0.099	3262.202	296811	11.11	0.000	80.570	10659	96.30	0.000	23.524	5136	96.30
	All	0.000	3.964	3246	98.77	0.000	3.866	2690	98.77	0.085	3407.006	612120	8.64	0.000	47.608	7128	98.77	0.000	23.633	3400	98.77
1.5	4	0.000	5.090	3822	100.00	0.000	3.603	2587	100.00	0.077	3600.198	315071	0.00	0.000	102.199	9880	100.00	0.000	74.187	7854	100.00
	5	0.000	21.900	14434	100.00	0.000	53.649	13845	100.00	0.108	3600.500	271807	0.00	0.000	399.755	32629	100.00	0.000	258.673	20341	100.00
	6	0.001	201.818	60720	92.59	0.001	192.653	51753	96.30	0.140	3600.264	330030	0.00	0.004	611.300	52744	96.30	0.001	451.479	37789	96.30
	All	0.000	76.269	26325	97.53	0.000	83.302	22728	98.77	0.109	3600.321	305636	0.00	0.001	371.085	31751	98.77	0.000	261.446	21994	98.77
2.0	4	0.000	18.679	19375	100.00	0.000	17.139	8937	100.00	0.075	3600.254	208456	0.00	0.000	395.423	33282	100.00	0.000	253.997	20417	100.00
	5	0.000	145.379	79703	100.00	0.000	139.415	80968	100.00	0.120	3600.208	305714	0.00	0.007	1338.818	90529	88.89	0.004	992.724	67959	85.19
	6	0.001	292.857	111567	92.59	0.000	230.131	89261	96.30	0.145	3466.892	146830	0.00	0.010	1735.771	133800	70.37	0.006	1540.471	75724	74.07
	All	0.000	152.305	70215	97.53	0.000	128.895	59722	98.77	0.113	3555.785	220333	0.00	0.005	1156.671	85870	86.42	0.003	929.064	54700	86.42
All		0.000	77.513	33262	97.94	0.000	72.021	28380	98.77	0.102	3521.037	379363	2.88	0.002	525.121	41583	94.65	0.001	404.714	26698	94.65

Table 5: Comparison of base and strengthened models when solving the MIP formulations **M1**, **M2** and **M3** with solver `Gurobi`.

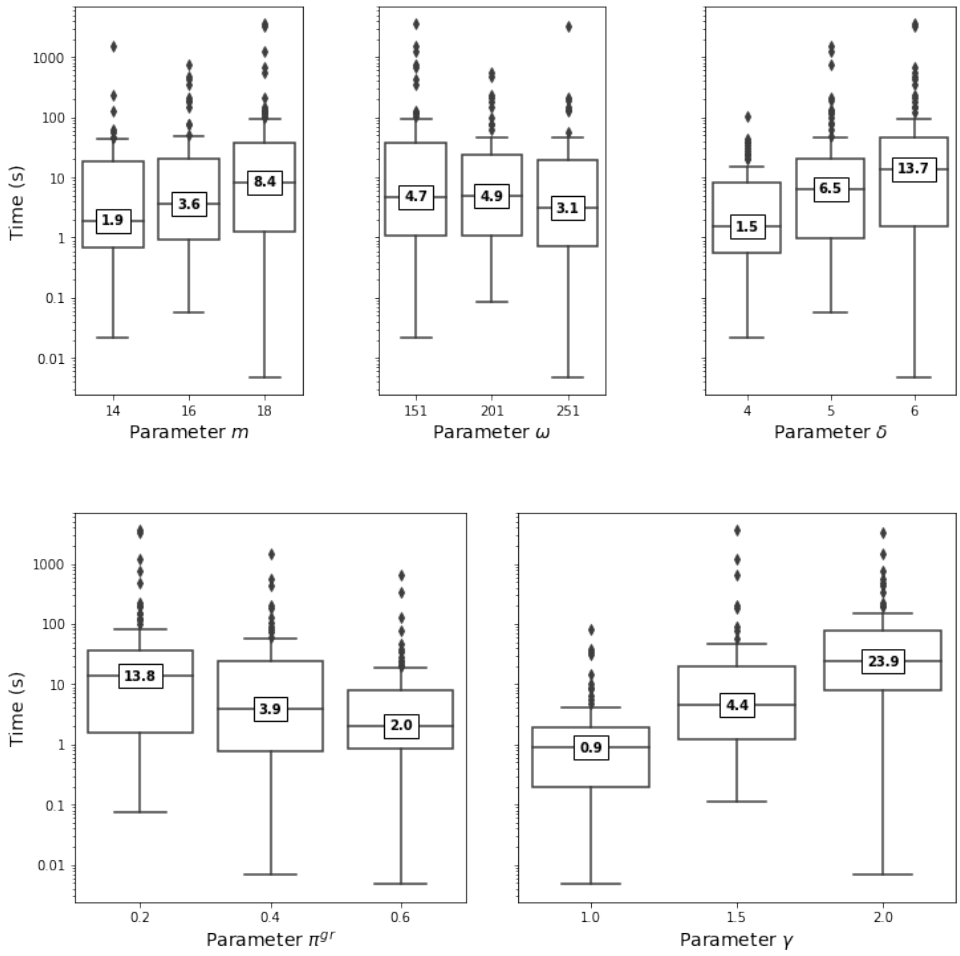

Figure 6: Impact of instance generation parameters on the runtime of model **M1** + TaskInc + MinIT.

solving more than 94% of the instances. Its average runtime and gap are worse than those of **M1** over all subsets of instances and, therefore, we see no reason to prefer it to **M1** even on a subset of the instances. Finally, formulation **M2** is the weakest of the three. It starts with the weakest continuous relaxation at the root node and shows the highest gaps and the lowest number of closed instances at the end of the 1-hour time limit. Its average gaps, however, are always under 0.2%, showing that **M2** suffers from a "tail effect": the dual bound increases quickly in the beginning, but then stops at a sub-optimal value. We also point out that additional experiments based on a dataset that was not generated with backbone instances (i.e., in which the instances were not necessarily feasible) showed that **M2** was the fastest model to detect infeasibility. In practical applications, **M2** could cover a complementary role to **M1**; for example, a decision-maker can run both models in parallel when it is not certain that a feasible elevator schedule exists.

7.1.3 Impact of instance generation parameters

We investigate which instance characteristics are associated with harder-to-solve problems. To this end, we use the best-performing model identified in Section 7.1.2, **M1** + TaskInc + MinIT, and analyse how its runtime varies depending on parameters m, ω, δ, π^{gr}, and γ.

Figure 6 shows how average runtimes vary with the above parameters. Each plot corresponds to a different parameter and each box to a different value of that parameter. The average runtimes over all Synthetic instances are reported on the y axis, on a logarithmic scale. The horizontal lines inside the boxes and the numbers over them represent the median runtimes. Boxes span from the 1st to the 3rd quartiles; whiskers extend to the rest of the distributions, except for outliers marked with diamond flyers. A point is deemed an outlier if it lies more than 1.5 times the inter-quartile range outside the two central quartiles.

All instance generation parameters have a considerable impact on the solver runtime. Larger instances with more trays (higher values of m) are, predictably, harder to solve. Instances in which jobs are "squeezed" in a small time horizon (small ω) and instances with longer shelf lives

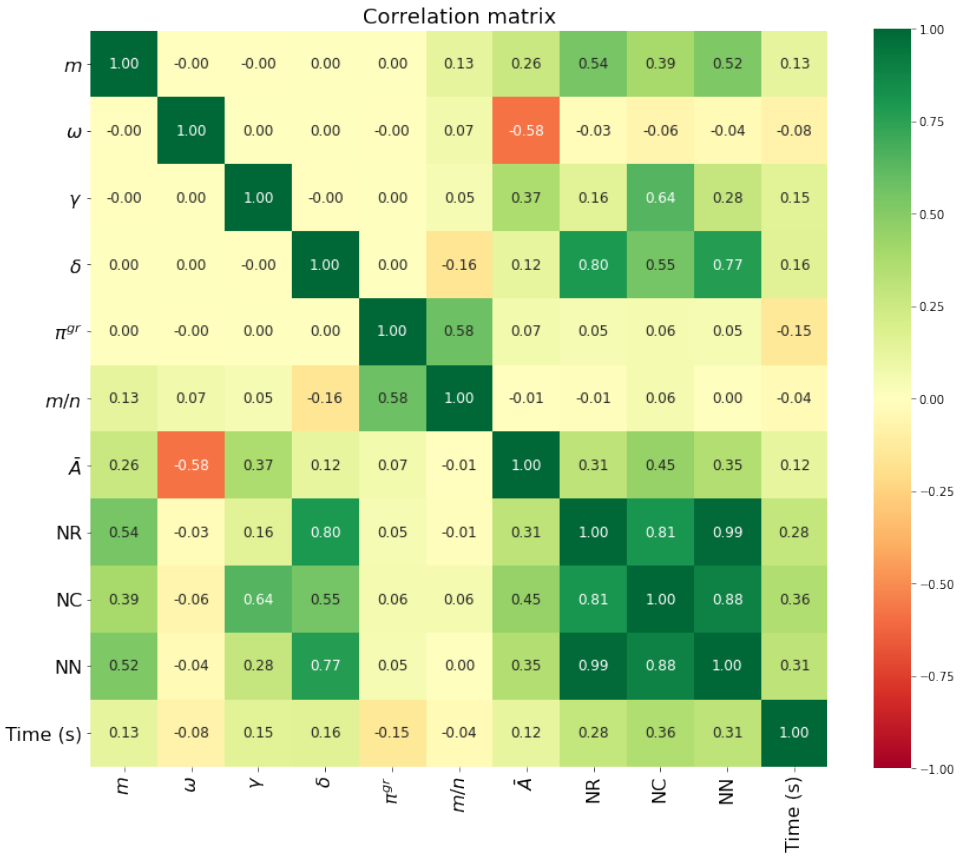

Figure 7: Correlation matrix between instance generation parameters and key indicators related to the instances (the ratio of trays to shelves, m/n, and the average number of open time windows per instant $\bar{A}$) and to model **M1** + TaskInc (the number of rows NR, of columns NC, of non-zeroes NN, and the runtime in seconds).

(smaller values of the greedy probability parameter π^{gr}) also correspond to longer runtimes. In general, instances with more overlap of task time windows are harder to solve, as confirmed by the large increase in runtimes when increasing the time-window half-width parameter γ. Increasing the number of tasks assigned to each tray (parameter δ) also makes the model more challenging.

To better study the relationship between instance and MIP model characteristics, we report in Figure 7 the correlation matrix between:

- The instance generation parameters: m, ω, δ, π^{gr}, and γ.
- Two indicators which measure how "dense" an instance is. The first is the number of trays per shelf (m/n). The second is the average number of open time windows per time instant, defined as
$$\bar{A} = \frac{1}{\omega} \sum_{t \in T} \sum_{j \in J'_t} (\beta_{tj} - \alpha_{tj}).$$
- Three indicators of the size of the resulting MIP: the number of columns (NC), the number of rows (NR, which includes both rows in the basic formulation of **M1** and valid inequalities), and the number of non-zero entries in the constraint matrix (NN).
- The runtime in seconds.

Parameters m and δ have the largest impact on the size of the model, because they directly affect the size of set V, i.e., the number of y variables. Indicator $\bar{A}$ also affects the size of the model because the more time windows overlap, the fewer tuples (t_1, j_1, t_2, j_2) can be excluded from set V using the procedure described in Section 5.1. Larger models (higher NR, NC, NN) result in longer runtimes, while indicator m/n negatively correlates with runtimes. This is because, everything else being equal, in instances with larger m/n the same number of trays is packed in fewer shelves during the instance generation procedure. This is possible because tray shelf lives are short and non-overlapping, which are conditions which make fixing variables Δ and y easier, and allow to generate more inequalities TaskInc.

7.1.4 Comparison between Branch-and-bound and Constraint Programming

We end this section by comparing the MIP and CP approaches for the VFEEMP. To this end, we compare the best MIP-based model (**M1** + TaskInc + MinTT) with constraint-programming model **M4**.

		M1 + TaskInc + MinIt			**M4**		
γ	δ	Gap%	T	Opt%	Gap%	T	Opt%
1.0	2	0.000	2.623	100.00	0.586	3600.000	0.00
	3	0.000	2.937	100.00	0.433	3600.000	0.00
	4	0.000	6.037	100.00	0.578	3466.667	3.70
	All	0.000	3.866	100.00	0.532	3555.556	1.23
1.5	2	0.000	3.603	100.00	0.800	3600.000	0.00
	3	0.000	53.649	100.00	1.624	3600.000	0.00
	4	0.012	192.651	96.30	2.046	3600.000	0.00
	All	0.004	83.301	98.77	1.490	3600.000	0.00
2.0	2	0.000	17.139	100.00	1.452	3600.000	0.00
	3	0.000	139.415	100.00	2.771	3600.000	0.00
	4	0.000	230.131	100.00	2.853	3466.667	3.70
	All	0.000	128.895	100.00	2.352	3555.556	1.23
All		0.001	72.020	99.59	1.458	3570.371	0.82

Table 6: Comparison of the best MIP model, **M1** + TaskInc + MinTT, and the CP model, **M4**.

Table 6 reports the results of this experiment. To compare the primal solutions from MIP and CP algorithms, columns "Gap%" refer to the percentage gap with the optimal solution of each instance (or the best-known solution for the only instance in the Synthetic set which we could not solve to optimality). We report the run-times in seconds in column "T". Column "Opt%" lists the percentage of instances for which the final solution returned by the algorithm was equal to the optimal solution. This does not mean that **CP** could *prove* that the solution was optimal: indeed, except for two small instances, **CP** could never prove optimality within the time limit. Even though **CP** is not competitive with model **M1**, its optimality gaps are small (around 1.5% on average). Further analysis on **CP** outputs showed that the best solution was usually found in the first five minutes. This indicates that, for very difficult instances, there could be an interest in running **CP** for a limited amount of time to find an initial solution and use it as a warm-start in the other models.

7.2 Results on the Realistic instances

This section reports results on the Realistic instances, which differ from the Synthetic ones due to their much larger time horizons and the higher number of tasks assigned to each tray. During preliminary experiments, we noticed that MIP-based models often struggled to find any primal solution. For this reason, during these experiments, we provided these models with an initial feasible solution obtained running the CP model **M4** for 5 minutes (thus leaving 55 minutes as available runtime for the MIP models).

To account for the larger instance size, we increased the amount of allocated RAM from 4GB to 12GB. Even with this adjustment, though, models **M2** and **M3** ran out of memory on, respectively, 18 and 19 of the 50 instances. Therefore, we report results comparing models **M1** + TaskInc + MinIT and **M4**, i.e., on the two models that solved all instances in the set. Moreover, model **M2** proved that one of the instances we were provided with was infeasible. Thus, the following results refer to the remaining 49 instances.

Table 7 reports the results of our experiments. Because the CP algorithm does not provide meaningful gaps and because we do not know the optimal solution to most of the Realistic instances, to obtain a uniform comparison we compute the values in columns "Gap%" as the percentage gap between the primal bound provided by each of the two algorithms and the dual bound provided by **M1**. Columns "T" report the average time in seconds. Each row aggregates

		M1$^+$		M4	
n	#	Gap%	T	Gap%	T
5	8	6.07	2275.21	24.05	3600.00
10	15	35.67	3600.00	42.36	3600.00
15	14	61.76	3600.00	57.31	3600.00
20	12	67.97	3600.00	60.57	3600.00
All		46.20	3385.49	47.84	3600.00

Table 7: Results of models **M1** + TaskInc + MinTT and **M4** on the Realistic instances.

results over all instances with the same number of shelves, whose number is reported in column "#".

We note that the Realistic instances are hard to solve to optimality. Percentage gaps at the end of the time limit stay high, although it is not clear whether this is due to a low quality primal bound or a low quality dual bound (or both). We observe that for the largest instances ($n \geq 15$), the primal bound found by the CP model was better than the one found by **M1**. Further analysis showed that for these instances, the CP model improved its best solution consistently until the end of the time limit, which is a major difference with respect to the Synthetic instances where the best incumbent was found within the first five minutes. We also note that the size of model **M1**$^+$ was huge (around 35000 variables and 3.25 million constraints on average), mainly because of the valid inequalities. As we observed that most constraints were removed by the inner preprocessing of `Gurobi`, we tried the simplest version of **M1** (i.e., without any valid inequalities), but the results were significantly worse.

8 Conclusions and future research

We introduced the Vertical Farming Elevator Energy Minimisation Problem (VFEEMP), a real-world problem arising from the operations of autonomous vertical farms. We showed that its decision version is NP-complete and presented three MIP formulations together with a set of valid inequalities, and a CP model for the problem. We also introduced two sets of benchmark instances, one derived from real-life data, and the other to determine the instance characteristics that make the problem harder to solve. We empirically evaluated the performance of the models and the impact of the valid inequalities and showed that (i) MIP model **M1** with valid inequalities TaskInc and MinIT obtained the best performance on synthetic and small-size realistic instances, and (ii) CP model **M4** helps finding good quality initial solutions for small-size synthetic instances and was superior to **M1** for large-size realistic instances. We also identified parameters that make instances harder to solve for our models: a large number of trays, a large number of tasks per tray, and large time windows. Indeed, the two first characteristics directly influence the number of variables and constraints in the models, while the latter increase the number of possible quadruple (t_1, j_1, t_2, j_2) that are valid indices for variables y.

Future work shall focus on the development of specialised heuristics for the problem. It would also be interesting to study the development of decomposition approaches. For example, disregarding time window violations in the master problem and checking the validity of the resulting schedule in the slave problem. Another possibly fruitful research avenue is time decomposition. For example, in Figure 5, solving the instance first for trays 1,2,3,5, and the first tasks of tray 4, and then tackling the remaining tasks of tray 4 and tray 6. However, since the windows are relatives, such decomposition would not necessarily reach an optimal solution.

Furthermore, other relevant real-world variants of the problem could be studied: in a generalisation of our problem, the planner could be allowed to decide the shelf associated with each tray. In other words, the control system of the VF tower would have to jointly assign trays to shelves and plan the movements of the elevator. In a further generalisation, we could allow the elevator to move a tray from one shelf to another (empty) shelf at any point during the growth period of the tray. Even though this operation has some energy costs, it can potentially bring more savings on the long run, for example, when bringing an isolated tray closer to the rest of the occupied shelves.

Acknowledgements

Alberto Santini was partially funded by: MICINN (Spain) through the programme *Juan de la Cierva Formación*; AEI (Spain) and the Barcelona Graduate School of Economics (Spain) through *Severo Ochoa* grant CEX2019-000915-S; the European Union's Horizon 2020 research and innovation programme under a *Marie Skłodowska-Curie* grant (EUTOPIA Cofund); ESSEC Business School (France) through the *Visiting Professor* programme.

References

[1] Rami Abukhader and Samer Kakoore. "Artificial intelligence for vertical farming. Controlling the food production". MA thesis. Mälardalen University, 2021. URL: https://www.diva-portal.org/smash/get/diva2:1526309/FULLTEXT01.pdf.

[2] Dafni Despoina Avgoustaki and George Xydis. "Indoor vertical farming in the urban nexus context: business growth and resource savings". In: *Sustainability* 12.5 (2020). DOI: 10.3390/su12051965.

[3] Andrew Beacham, Laura Vickers, and James Monaghan. "Vertical farming: a summary of approaches to growing skywards". In: *The Journal of Horticultural Science and Biotechnology* 94.3 (2019), pp. 277–283. DOI: 10.1080/14620316.2019.1574214.

[4] Kurt Benke and Bruce Tomkins. "Future food-production systems: vertical farming and controlled-environment agriculture". In: *Sustainability: Science, Practice and Policy* 13.1 (2017), pp. 13–26. DOI: 10.1080/15487733.2017.1394054.

[5] Julia Bennell, Toni Martinez, and Chris Potts. "Scheduling for the growing of crops to meet demand". In: *Mic'17. Session T2.2: Scheduling*. Proceeding of the 12th Metaheuristics International Conference (Universitat Pompeu Fabra). Barcelona, Spain, 2017. URL: https://web.archive.org/web/20210116182958/https://easychair.org/smart-program/MIC'2017/2017-07-06.html. (archived: 2021-01-16).

[6] Shaylin Cetegen and Matthew Stuber. "Optimal design of controlled environment agricultural systems under market uncertainty". In: *Computers & Chemical Engineering* 149 (2021). DOI: 10.1016/j.compchemeng.2021.107285.

[7] Malek Al-Chalabi. "Vertical farming: skyscraper sustainability?" In: *Sustainable Cities and Society* 18 (2015), pp. 74–77. ISSN: 2210-6707. DOI: 10.1016/j.scs.2015.06.003.

[8] Nina Fedoroff. "Food in a future of 10 billion". In: *Agriculture & Food Security* 4.1 (2015), pp. 1–10. DOI: 10.1186/s40066-015-0031-7.

[9] Intelligent Growth Solutions, Ltd. *IGS growth towers*. 2021. URL: https://web.archive.org/web/20210221183103/https://www.intelligentgrowth.io/technology/growth-towers. (archived: 2021-02-21).

[10] Stefan Irnich and Daniel Villeneuve. "The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$". In: *INFORMS Journal on Computing* 18.3 (2006), pp. 391–406. DOI: 10.1287/ijoc.1040.0117.

[11] Imdat Kara and Tusan Derya. "Formulations for minimizing tour duration of the traveling salesman problem with time windows". In: *Procedia Economics and Finance* 26 (2015), pp. 1026–1034. DOI: 10.1016/s2212-5671(15)00926-0.

[12] Toyoki Kozai. *Smart plant factory: the next generation indoor vertical farms*. Springer, 2018. DOI: 10.1007/978-981-13-1065-2.

[13] Toyoki Kozai, Genhua Niu, and Michiko Takagaki. *Plant factory: an indoor vertical farming system for efficient quality food production*. Elsevier Academic Press, 2015. DOI: 10.1016/c2014-0-01039-8.

[14] Ferdinando Pezzella, Gianluca Morganti, and Gianfranco Ciaschetti. "A genetic algorithm for the flexible job-shop scheduling problem". In: *Computers & Operations Research* 35 (10 2008), pp. 3202–3212. DOI: 10.1016/j.cor.2007.02.014.

[15] Yuzhuo Qiu, Jun Qiao, and Panos Pardalos. "A branch-and-price algorithm for production routing problems with carbon cap-and-trade". In: *Omega* 68 (2017), pp. 49–61. DOI: 10.1016/j.omega.2016.06.001.

[16] Alberto Santini. *energy-efficient-automatic-vertical-farms*. Aug. 2021. DOI: 10.5281/zenodo.5185902. URL: https://github.com/alberto-santini/energy-efficient-vertical-farms.

[17] Alberto Santini, Enrico Bartolini, Michael Schneider, and Vinicius Greco de Lemos. "The crop growth planning problem in vertical farming". In: *European Journal of Operational Research* 294 (1 2021), pp. 377–390. DOI: 10.1016/j.ejor.2021.01.034.

[18] Alberto Santini, Henrik Alsing Friberg, and Stefan Ropke. "A note on a model for quay crane scheduling with non-crossing constraints". In: *Engineering Optimization* 47.6 (2015), pp. 860–865. DOI: 10.1080/0305215x.2014.958731.

[19] Xueqi Wu and Ada Che. "Energy-efficient no-wait permutation flow shop scheduling by adaptive multi-objective variable neighborhood search". In: *Omega* 94 (2020). DOI: 10.1016/j.omega.2019.102117.

[20] Chao-Lung Yang, Yulius Hari, and Yan-Fu Kuo. "Multiple-crop scheduling for plant factory". In: *Ismab'12*. Proceedings of the 6th International Symposium on Machinery and Mechatronics for Agriculture and Biosystems Engineering (June 18–20, 2012). Jeonju, Korea, 2012.

[21] Chao-Lung Yang, Sin-Jie Huang, and Chit-Hui Ang. "Recursive heuristic scheduling method for multi-crop plant factory with solar panel roof". In: *Computers and Electronics in Agriculture* 165 (2019). DOI: 10.1016/j.compag.2019.104941.