Bachelor Thesis

Salima Jaoua, Dana Kalaaji

Chair of Discrete Optimization

GrowBotHub's scheduling

Spring 2021

Thesis Advisor and Head of chair
Prof. Friedrich Eisenbrand
EPFL / DISOPT

Supervisor and PHD student
Jonas Racine
EPFL / DISOPT

Acknowledgment

We want to thank our thesis supervisor Jonas Racine who has available to help us during all this semester and gave us precious advice and feedback. We also want to thank Aurélien Debbas, GrowBotHub's president with whom we had a weekly reunion to answer our questions.

Contents

1 Introduction **4**
1.1 GrowBotHub . . . 4
1.2 Scheduling Team . . . 5

2 Some Background And Our Goals **5**
2.1 What Has Been Done Before Us ? . . . 5
2.1.1 Modelization Of The Problem . . . 5
2.2 Our Goals This Semester . . . 8

3 Time Analysis **9**
3.1 Hypothetical Data . . . 9
3.2 Real Data . . . 9
3.3 Modification of the Graph . . . 9

4 Optimization of the Problem **10**
4.1 Path Formulation . . . 10
4.2 Column Generation . . . 11

5 Implementing Constraints **13**
5.1 Diversity . . . 13
5.1.1 Harvesting Different Species of Plants Each Day . . . 13
5.1.2 Harvesting At Least One Plant Each Day . . . 14
5.1.3 Analysis . . . 14
5.1.4 Alternative Solution . . . 15
5.2 Plant's size . . . 15
5.2.1 Changing The Graph's Architecture . . . 15
5.2.2 Implementing The Constraint . . . 16
5.3 Analysis . . . 17

6 Note On The Strawberries **17**

7 Figuring Out The Optimal Number Of Modules **18**

8 Conclusion **18**
8.1 Next Year's Project . . . 19

9 Annex **21**
9.1 The inputs . . . 21
9.2 The graph . . . 21
9.3 Extract of the "instruction" file . . . 22
9.4 Extract of the "status of a module" file . . . 22
9.5 Extract of the "harvesting data" file . . . 23

1 Introduction

1.1 GrowBotHub

This report concludes our semester project done within GrowBotHub. The latter is an association at EPFL that is exploring agricultural solutions for space exploration. Our focus is on designing and building a fully automated system for growing and picking-up plants in a vertical farm in space, particularly on the moon. As a participant of the interdisciplinary project "IGLUNA: A Habitat in Ice", we will get to test a prototype of the system at IGLUNA's 2021 Field Campaign next summer.

GrowBotHub's vertical farm' structure (shown in Figure 1) consists of multiple shelves, each containing multiple growth modules with 5 holes; and a robotic arm that moves, harvests, and seeds the plants. The vegetables are grown using an aeroponic system (shown in Figure 2) which consists in feeding the roots of vegetables by sprinkling water containing appropriate nutrients. Thus, not all modules are identical as they contains specific nutrients. This method is used because plants grow faster and it allows us to reduce water and energy consumption compared to other techniques. To fully grow, a plant needs different nutrients at different moments of its growth phase and thus has to be moved between modules at specific points in time. There lies the scheduling problem our semester project will tackle.

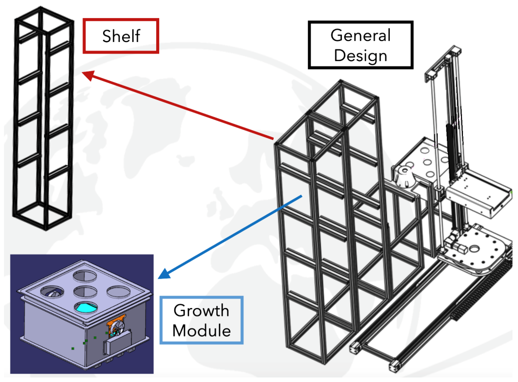


Figure 1: Farm's structure

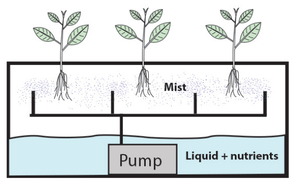


Figure 2: Growth module with 3 holes using the aeroponic method to grow plants

The particularity of our bachelor thesis lies in the fact that it is one part of a big project, the latter is implemented in parallel by 5 different teams:

- Structure: focuses on designing the shelves, the growth modules, displacement of the robot and the electric connections
- Scheduling: our team, focuses the algorithm for the movements of vegetables across the different growth modules
- Aeroponics: focuses on planning the nutrients to be used when watering the plants, more details will be explained in the next subsection
- Robotics : focuses on all the programs for the different movements of the robotic arm, and on 3D data vision to verify the growth of vegetables before moving or harvesting them.
- Networking : focuses on all the communication systems between the different components of the project and a remote control from earth.
- Chemical-engineering : focuses the different nutrients to satisfy a plant's needs, and the use of waste.

1.2 Scheduling Team

The aim of the scheduling team is to develop an algorithm that dictates the movements of plants across the farm in order to produce a maximum number of plants in a given period of time while respecting some constraints given by GrowBotHub. It also has to give the different instructions that the robotic arm will have to carry out. In addition to that, we will address the problem of re-scheduling, which consists of computing a new schedule on the spot in the case of an unexpected event happening while executing the original one (for example, if a specific plant is taking too much time to grow and needs to stay in a certain growth module longer than expected).

2 Some Background And Our Goals

2.1 What Has Been Done Before Us ?

Our project is the continuity of the project carried out last semester by the previous scheduling team (Nicolas Vial and Augustin Henry). They were able to come up with a algorithm coded in python that uses the open source Pulp solver. It takes as input

- the number of growth modules
- the number of holes per module, which is 5
- the different types of plants and their properties (total growth time, number of days spent in each growth module, size of the plant over time. . .)
- the time over which we want to schedule, we call this number `Horizon`
- the distances between the holes of a module

and outputs

- a text file containing the instruction day per day for the robotic arm
- a text file containing the status of each growth module: for each module and for each day, it tells us what plants are contained in its holes
- a png image of the graph given to the solver
- a png image of the graph outputted by the solver

2.1.1 Modelization Of The Problem

Last semester the problem was modeled as a multi-commodity flow problem where each node stands for a hole of a growth module. In our example below we consider that we have two module with one hole each as shown in Figure 3. The graph is a time expanded one so, for each unit of time, we have one copy of each node. Figure 4 is a graph where `Horizon` is 3 days. The left-most node stands for a hole on day 0, the one on its right for a hole on day 1 and so on. Also, for each type of plant we have one source and one sink shown respectively in Figures 5 and 6 where we consider 2 species of plants.

While there is enough time for a plant to grow, each source creates and connects an edge to the graph for each unit of time (days in our case). For example, imagine the plant whose source is represented by the pink node takes 2 days to grow, then if `Horizon` is 3 days, it will have enough time to grow it if we plant a seed of day 0 and day 1 but not on day 2. This is shown in Figure 7 by having an edge from the pink plant's source to a node on day 0 and day 1 but no edge connecting the source to a node on day 2. Figure 8 shows the completed graph where its edges stand for possible movements of the plants. Then, we give this graph as input to a solver that outputs the solution shown in Figure 9. The darker edges are the ones used by the solution, and the flow of each edge represents the number of plants carried by it.

Figure 3: 2 modules with 1 node each

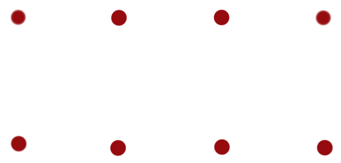


Figure 4: `Horizon` is 3 days

Figure 5: 2 species of plants, hence 2 sources

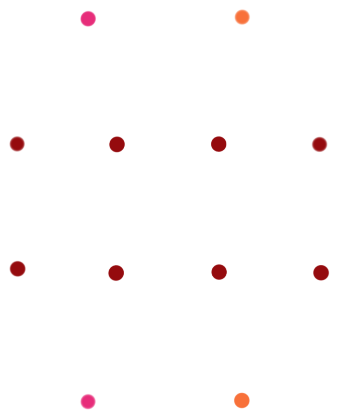


Figure 6: 2 species of plants, hence 2 sinks

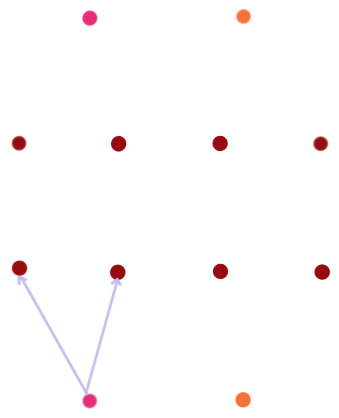


Figure 7: For each plant if possible we connect an edge to the graph

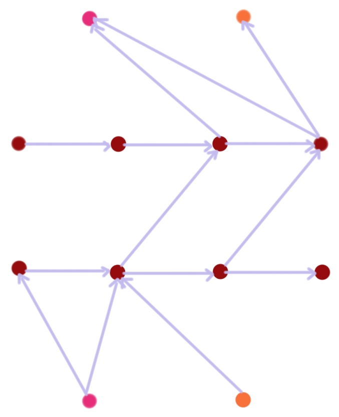


Figure 8: Completed graph given to the solver

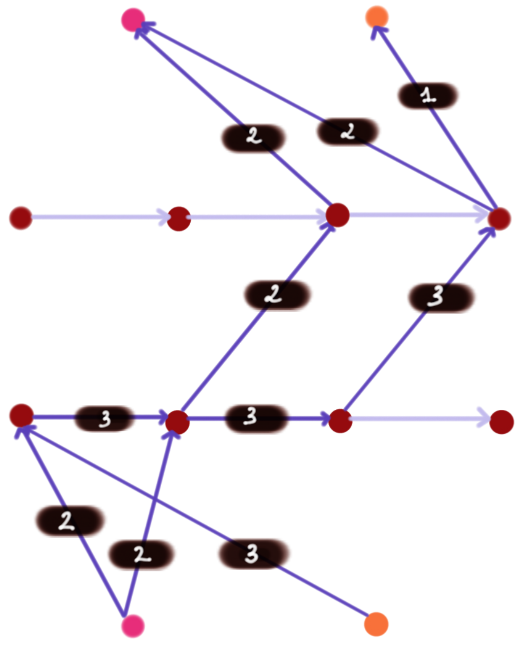


Figure 9: Graph outputted by the solver

To summarize this, we get a graph where each node stands for a hole of a module on a specific day and the edges represent the possible movements of the plants between holes. We also add a source s and a sink t for each type of plant so that an edge that connect s to the graph represents the action of seeding a plant from this specie. And the edges that connect the graph to the sink t represents the action of harvesting a plant from this specie.

With this representation, we can force certain type of plant to follow a path. To know which type of plant is going through a specific edge and when this plant has been planted, they introduced a multi-commodity flow. Each source creates one commodity in every edge that connects it to the graph. This means that the source creates its own flow. Therefore, we can have different flows going through a unique edge. For example, if an edge represents a road, then we can have a unique flow that represents vehicles in general, or if we want to be more precise, we can have 3 different flows to represent trucks, motorcycles and cars.

Moreover, and like all flow networks, each edge has a certain capacity. But in the case of a multi-commodity graph, since each edge contains multiple commodities (hence multiple flows), each one of these commodities also has a capacity. The sum of all these capacities has to respect the overall capacity of the edge. To represent the flow that goes through this graph, they decided to creates one variable per edge per commodity:

$$x_{ijk} \quad \forall (i,j) \in E \quad \forall k \in K$$

Their optimization problem is now :

$$\text{maximize} \sum_{s}^{S} \sum_{(i,j)\in\delta^+(s)} x_{ijk}$$

$$\text{subject to} \sum_{(i,j)\in\delta^+(n)} x_{ijk} - \sum_{(i,j)\in\delta^-(n)} x_{ijk} = 0, \forall k \in \{1....K\}, \forall n \in V \tag{1}$$

$$\forall (i,j) \in E, \forall k \in \{1...K\}, \sum_{k} x_{ijk} \leq 1, \tag{2}$$

$$\sum_{(i,j)\in\delta^+(n)} x_{ijk}\ growthRate_{ijk} + \sum_{(i,j)\in\delta^-(n)} x_{ijk}\ growthRate_{ijk} \leq sizeMax_{(n,h)}, \forall k \in \{1...K\} \tag{3}$$

$$\forall s \in S \sum_{(i,j)\in\delta^+(s)} x_{ijk} \geq \frac{\sum_{s}^{S} \sum_{(i,j)\in\delta^+(s)} x_{ijk}}{N} - \alpha, \alpha \geq 0, \forall k \in \{1...K\} \tag{4}$$

where V is our set of nodes, E the set of edges and K the number of commodities in the graph. $\delta^+(n)$ and $\delta^-(n)$ represent the incoming edges and the out-coming edges to a node n respectively.

In other words, we want to maximize the sink inflow while respecting certain restrictions. In the next points we will provide a detailed explanation on those four constraints:

(1) The sum of the flow entering all nodes (aside from the sources or the sinks) must be equal to the sum of the flow exiting from them.

(2) There must be at most one plant per hole at any time. We implement it by constraining the sum of all commodities on every edge to maximum one.

(3) Constraint of the plant's size. The two nodes n and h are neighbor holes. We get the current size by multiplying the size that the commodity k would have at the edge (i,j) (called `growthRate`) with the variable x_{ijk} which is either 1 or 0. So if a commodity is passing to that node, we would add its size to the sum, otherwise we add 0. The maximum size depends of the two neighbor nodes, it means that there is actually multiple possible distances between different holes in function of the design of the growth modules.

(4) We force the number of plants produced from each species to be close to the average, which is the total of plants produced divided by the number of different species. N is the number of different species and alpha is a manually set to a constant that is used to loosen up the constraint.

2.2 Our Goals This Semester

Our goal this semester is to

- Optimize what has been done last year.
- Implement additional features asked by the association.
- Implement rescheduling.

3 Time Analysis

3.1 Hypothetical Data

As said previously, our project is implemented in parallel by different teams. While last semester's scheduling team was designing their algorithm, the chemical-engineering team was still figuring out the growth time and nutrient requirements of all the different plants. So no data was available to the team, they were only able to test their code with hypothetical data they came up with. The team noticed that their running time was slow when running the algorithm with their hypothetical inputs, especially since they had to switch to the Pulp solver instead of the Gurobi one as the latter is faster but not open source.

3.2 Real Data

As of this semester, the chemical engineer team gave us all the data we needed. But we encountered a problem: last semester's hypothetical data was extremely far from the reality. For example, they imagined that a strawberry takes 5 days to grow, when in reality it takes approximately 120 to be completely developed. They seem to have tested their codes for only small values of `Horizon` whereas we want to test it for 180 days (6 months). These huge gap between the sizes they considered and the real data forced us to change some parts of the code.

First, to avoid having two plant crashing into each other, they implemented a constraint on the plant's size. So when giving the inputs, we need to enter the size of each plant for every unit of time. This is quick when the plant takes 5 days to grow. But when its growth time is 120 days it is more tedious and error prone, especially since most of these numbers are going to be identical.

Another example is the fact that they thought that each plant would have very specific needs that differs from other plants, hence no two module would be of the same type. Note a type of module is defined by the nutrients it contains. In reality we only have 4 different type of modules: seedling, vegetative growth, flowering and development that we will denote as module of type 1, 2, 3 and 4 respectively for the rest of the report. Moreover, all the plants go through these modules in the same order (for example, they all start at the module "seedling" but the duration of their stay varies from plant to plant). However not all plants go through all modules: most of the plants finish their growth in module 2, thus they won't go to module 3 and 4.

Once we made these little modification in the code, we were able to run it with the real inputs. But we encountered a bigger problem: the algorithm crashes after a many hours (more tham 12 hours) with a MemoryError, because the real inputs are too big. The problem persists even if we tried to acceptably reduce the sizes of the inputs: the resulting graph is till too large, it was using more than 90% of our computer's memory. We need to optimize the problem.

3.3 Modification of the Graph

As we saw in the section 3.2, the graph is too large. So we tried to reduce the size of the graph without losing too much information. What if each node represented a type of module instead of a hole ? The number of edges will decrease since the number of nodes decreases considerably. We would have to modify the capacities of the edges since we are no longer moving one plant from a hole to another, we are going to relocate 5 plants at most to a growth module to another. Hence, the capacity of each edge becomes the number of types of growth module multiplied by 5.

and Figure 12 display the difference between the size of the graph before and after the graph's change of architecture. The total number of edges is 35 time less than before: it went from 106 240 to 3010.

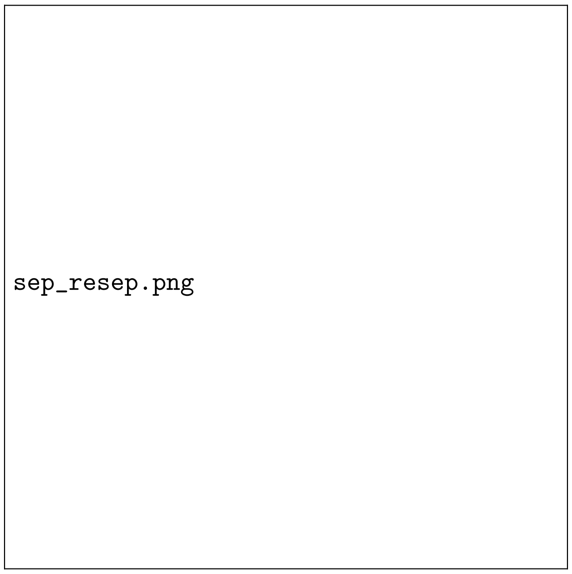


Figure 10: Enter Caption

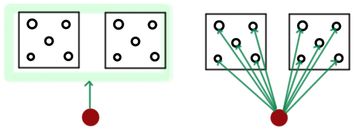
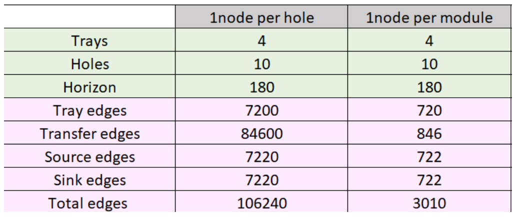


Figure 11: Comparison of the two different architectures. Left picture is a graph where 1 node stands for a type of module. Right picture is a graph where 1 node stands for a hole of a module.



	1node per hole	1node per module
Trays	4	4
Holes	10	10
Horizon	180	180
Tray edges	7200	720
Transfer edges	84600	846
Source edges	7220	722
Sink edges	7220	722
Total edges	106240	3010

Figure 12: Number of each type of edges for the two types of graphs

We decided to proceed to this graph's modification. Now when we run our algorithm with the real data it takes approximately 2 minutes to find a solution that is 95 % optimal. We don't run the code to find its 100% optimal solution because the solver finds that there is an optimal solution that is not reached but takes hours and sometimes loops to find it. We noticed that the loss of plants compared to the 95 % optimal solution is 1 to 2 plants, which is negligible for GrowBotHub.

4 Optimization of the Problem

Note: We decided to not implement the column generation method in our algorithm because the running time is good enough and we were lacking time. However we kept this method in our report in case GrowBotHub adds other constraints in the future that negatively impact the running time.

The problem with our solution is that we completely loose the control on the plants' position within a growth module, thus we have no way of controlling their sizes to prevent them from crushing one another. Therefore we are going to see if instead of changing the architecture of our graph (or in addition to that), we can use an optimisation method to decrease our running time.

4.1 Path Formulation

First we are going to reformulate the problem into a path flow problem. The reformulation is indeed help because note that:

- First of all, we don't have a multi-commodity flow anymore.
- Note that, in the graph, we have at most one path per day per plant; this means that for example, in 30 days, and with 6 plants (the real number of plants that we have in disposition), we have $30 \times 6 = 180$ paths possible. Or before, we had a variable x_{ijk} defined for every edge and every commodity.
- This reformulation leads us to think of two methods of optimization.
  - Column generation method
  - Maximum independent set.

For now, we are going to ignore the constraint on the plant's size. We will restate our problem using path flow.
We denote :

- $\forall i \in \{1 \dots N\} : \mathcal{P}_i = \{\text{simple path from} s_i \text{ to } t_i\}$
  where s_i and t_i corresponds to the source and the sink for the type of plant i. Note that we have N type of plants, therefore we have N sources and N sinks.
- $I = \{1 \dots N\}$, the set of species of plants.
- $\mathcal{P} = \{\mathcal{P}_1 \bigcup \cdots \bigcup \mathcal{P}_N\}$
- $\forall e \in E : \mathcal{P}_e = \{p \in \mathcal{P} : e \in p\}$
  where E denotes the set of edges in the graph, and we say that e is in p if and only if the path p goes through the edge e.
- the flow that goes thought the path p is going to be denotes by $f(p)$

With all this notation, we can then rewrite our model. The problem becomes :

$$\text{maximize} \sum_{p \in \mathcal{P}} f(p) \tag{5}$$

$$\text{subject to} \sum_{p \in P^e} f(p) \leq 1 \quad \forall e \in E$$

$$\sum_{p \in P_i} f(p) \quad \geq \frac{\sum\limits_{p \in P} f(p)}{N} - \alpha, \quad \alpha \geq 0, \forall i \in I \tag{6}$$

$$f(p) \geq 0 \tag{7}$$

4.2 Column Generation

We decided to opt for the Column generation method. It is a method to efficiently solve linear programs with a huge number of variables. It consists of solving a restricted linear program which contains all constraints of the original program.
$\longrightarrow$ How does the column generation method works ?
We denote by (P) the foregoing problem with the path formulation. We denote its dual by (D).

We are not going to give the details of the computations but we get that :

$$\text{minimize} \sum_{e \in E} \lambda_e + \alpha N \sum_{i \in [I]} \mu_i$$

$$\begin{aligned}
\text{subject to} \sum_{e \in p} \lambda_e &\geq 1 \quad \forall p \in P \\
\sum_{i \in I - \{k\}} \mu_i - (1-N)\mu_k &\geq 1 \quad \forall p \in P_k \\
\lambda_e &\geq 0 \quad \forall e \in E \\
\lambda_k &\geq 0 \quad \forall k \in K
\end{aligned}$$

Now, we take $\mathcal{P}' \subseteq \mathcal{P}'$ a subset of all paths and we consider only the restricted problem $(P') \in (P)$. A feasible solution $(f'_p)_{p \in \mathcal{P}'}$ for (P') yields to a feasible solution for the problem (P) :

$$f_p = \begin{cases} f'_p \ \forall p \in \mathcal{P}' \\ \\ 0 \ \ \forall p \in \mathcal{P} \setminus \mathcal{P}' \end{cases}$$

We know that either :

- The solution is in fact optimal. In this case, we are done.
- Otherwise, when computing the dual, we have that this condition needs to be verified for all paths :
$$\sum_{e \in p} \lambda_e \geq 1 \quad \forall p \in P$$
If there exists a path p violating this condition, we know that the solution is not optimal. Therefore we add this path p in $\mathcal{P}'$ and repeat the process.

Theorem : The path formulation for the original problem can be solved in polynomial time.

This is good news if the number of variable is huge. But like mentioned before, the number of variable is in reality smaller than we expected. This comes from the fact that a type of plant can only take one path a day. But we can solve this issue. To do so, we add a different capacity for the variable of a path so that we take all the simple paths possible in the graph. Taking all the simple paths possible makes our number of variable growing exponentially. We will get a huge number of variable. Let's implement this :
When a path is not feasible for a plant, meaning that the plant can't follow the path to grow, we set the capacity of the flow going through this path to 0.

$$\text{maximize} \sum_{p \in \mathcal{P}} f(p)$$

$$\begin{aligned}
\text{subject to} \sum_{p \in \mathcal{P}^e} f(p) &\leq u_{e,p} \quad \forall e \in E \\
\sum_{p \in \mathcal{P}_i} f(p) \quad &\geq \frac{\sum\limits_{p \in \mathcal{P}} f(p)}{N} - \alpha, \quad \alpha \geq 0, \forall i \in I \\
f(p) &\geq 0
\end{aligned}$$

Where $u_{e,p} = \left\{ \begin{array}{ll} 1 & \text{if p is feasible} \\ 0 & \text{otherwise} \end{array} \right.$

Therefore, we can use the Column Generation method to have a better running time. Since for now, the running time is good, we let the implementation of the method for later.

5 Implementing Constraints

5.1 Diversity

After talking to the president of GrowBotHub, we were informed that some of the scheduling goals changed. At first, we only wanted to obtain an approximately equal number of plants at the end of the given time. Now we also want to obtain at least one plant every day, with different species each day. So our current diversity constraint needs to be updated

However, we decided to keep this constraint and implement a new one that respects the new goal. That way, if we successfully implement the new constraint, then the old one ensures that we still get a more general diversity. And if we don't succeed, the old one will at least contribute to having some sort of diversity. To implement this new constraint, we divide the task into two smaller ones: the first constraint will ensure that we harvest at least a plant each day and the second one will make sure that we harvest different species of plants each day.

5.1.1 Harvesting Different Species of Plants Each Day

To implement this new constraint, we need to diversify the outcomes each day. In order to do this, we are going to add a variable for every edge. Let's denote this variable $y_{ij}\ \ \forall (i,j) \in E$. We will use this variable to notify if they is a flow going thought the edge. Hence:

$$y_{ij} = \begin{cases} 1, & \text{if } \exists k \ \ x_{ijk} > 0 \ \forall (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

Where x_{ijk} correspond to the flow created by the commodity k going through the edge (i,j).

Now that we have an indicator of the flow, our idea was to limit the number of days where we harvest a certain type of plant over a given amount of time. For example, we are going to constrain the algorithm to harvest each species at most once over every 3 consecutive days:

$$\forall (i,j) \in E \ \ \forall p \ \ \forall d \ \ x_{ijk_{p_d}} \leq 5 y_{ij}$$

$$\forall i,j \ \ y_{ij} \in \{0,1\}$$

$$\forall d \ \ \forall p = 1....N \ \ \sum_{d}^{d+3} y_{t_{p_d}} \leq 1$$

The first condition ensures that y_{ij} is equal to 1 if the flow going through the edge (i,j) is non-null. The second one ensures that y is binary. The third one makes sure that for every day d and for all plant p the sum over the three consecutive days starting at d of $y_{t_{p_d}}$ must be less than 1. t_{p_d} corresponds to the edge that connect the sink of the plant p to the graph. We know that there is a unique edge of this type per day because the plant can only be harvested once a day. This is how we constructed our graph.

Note that here we randomly took 3 days, but it can be more or less. We will run the tests to see which number of days gives a good amount of plants while ensuring a good diversity.

5.1.2 Harvesting At Least One Plant Each Day

To implement this constraint, we create a new variable $(z_d)_{d\in\{1\dots D\}}$ for each day, where D is the number of day we have to plant. On a given day d, we have

$$z_d = \begin{cases} 0 & \text{if we harvest a plant on day d} \\ 1 & \text{otherwise} \end{cases}$$

The goal is to have $z_d = 0\ \forall d$ but this may be impossible. To maximize the number of days where we harvest a plant, we decided to introduce this constraint as a penalization. We subtract a penalty on each day where we don't harvest a plant : to do so, we subtract to the optimal value of our problem '$w \sum_{d=1}^{D} z_d$'. Where we try different values of w to see which one will minimize the most the number of days where we don't harvest a plant and maximize the number of plants in total.

5.1.3 Analysis

Now that we implemented these new constraints, we get a new linear program :

$$\begin{aligned}
\text{maximize}\quad & \sum_{p=1}^{P} \sum_{d=1}^{size(k_p)} \sum_{(i,j)\in\delta^+(s_p)} x_{ijk_{p_d}} - w\sum_{d=1}^{D} z_p - v \sum_{(i,j)\in E} y_{ij} \\
\text{subject to}\quad & \sum_{(i,j)\in\delta^+(n)} x_{ijk} - \sum_{(i,j)\in\delta^-(n)} x_{ijk} = 0 \quad \forall n \in V - \{S,T\} \\
& \forall (i,j)\in E \quad \sum_{p=1}^{N} \sum_{d=1}^{size(k_p)} x_{ijk_{p_d}} \le 5 \\
& \forall p \in \{1....N\} \quad \sum_{d=1}^{size(k_p)} \sum_{(i,j)\in\delta^+(s_p)} x_{ijk_{p_d}} \ge \frac{\sum_{p=1}^{N} \sum_{d=1}^{size(k_p)} \sum_{(i,j)\in\delta^+(s_p)} x_{ijk_{p_d}}}{N} - \alpha \quad \alpha \ge 0 \\
& z_d \ge 0 \\
& z_d \ge 1 - \sum_{p=1}^{N} x_{t_{p_d}} \\
& \forall (i,j)\in E \quad \forall p \quad \forall d \quad x_{ijk_{p_d}} \le 5y_{ij} \\
& \forall i,j \quad y_{ij} \in \{0,1\} \\
& \forall d \quad \forall p = 1....N \quad \sum_{d}^{d+3} y_{tp_d} \le 1 \\
& x_{ijk_{p_d}} \le 2
\end{aligned}$$

After implementing the first constraint and running the code, we noticed that we get a more even distribution of plants over time . However, the total number of plants changed, but not by a lot, we loose approximately 0 to 4 plants depending on the inputs. This loss is negligible. We also remarked that this constraint does not really slow down the running time: it still runs in less than 2 minutes. So overall the implementation was successful

The second constraint is more problematic as it really slows down our running time. Indeed, we went from approximately 2 minutes to more than 3 hours. Moreover, it turns out that if we limit the harvesting of a same type plants to at least every 2 days, the algorithm outputs only 2/3 of our previous total number of plants. Even worse, if we limit this to every 3 days, we get 0 plants.

The disadvantages of this constraint outweigh the advantages. When we showed these results to the association, they decided to drop it.

5.1.4 Alternative Solution

We then thought of others way to implement this problematic constraint and found one. A growth module can carry up to 5 plants, so we decided to establish the diversity of plants harvested each day within a growth module. To do so, we limit the number of plants of the same species to at most 2. This only changes the capacity per commodity of each edge. While testing the code we noticed that not only does this not slow down the running time, but it also does not change to overall number of plants.

The drawback of implementing this constraint this way is that the goal is not perfectly respected. We still have days where we harvest the same plants a few days in a row. But a compromise had to me made somewhere between a flawlessly respected constraint and good running time and number of plants. This solution is good enough for GrowBotHub.

5.2 Plant's size

Since we changed the architecture of the graph, the constraint on the plant size implemented last semester needs to be updated.

5.2.1 Changing The Graph's Architecture

The complication with our current representation of the problem is that we have no way of controlling the size of plants contained in each hole of a growth module as one node represents all the modules of a certain type. So, we can't know if there are two plants overlapping in a growth module, since we don't exactly know which plants are in which growth module. To overcome this we decided to change the architecture of the graph. We now have a graph where one node stands for a module as opposed to a type of module. Figure 13 shows how the architecture of the graph changed. At first, we had a graph where we one node stands for a hole, then we switched to a representation where we had one node for each type of module and now we are changing it to this representation where we have one nodes per module.

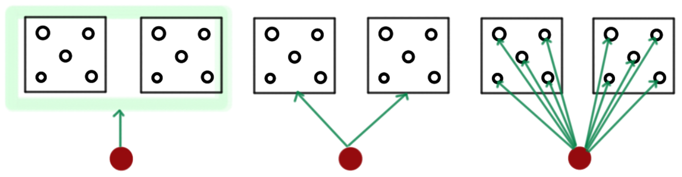


Figure 13: Comparison of the 3 different architecture. Left picture is a graph where 1 node stands for a type of module. Middle picture is a graph where 1 node stands for a module. Right picture is a graph where 1 node stands for a hole of a module

This allows us more control over what happens inside a module, and to know which plants is contains. When testing the updated code, we observed that the code does run on real sized inputs. However, the running time went from less than 2 minutes to approximately 3h30. Even thought the gap is large and that the code may seem to be slow, it actually still is a good running time since this algorithm is only meant to be ran once for the next 6 months.

5.2.2 Implementing The Constraint

To implement this constraint, the best scenario would be to know the size of each specie over time. And since we know the dimension of a module, this data would have helped us to come up with a precise way of implementing this size constraint ensuring that no two plants are overlapping. The problem is that no experiment has been done in the association. This means that we have no values of the plant's size. Since we have no data, we decided to detect which plants are the biggest by doing some researches in the internet. Knowing which one are considered as the biggest lets us know which one may cause problem over the time.

We found that 3 out of the 6 possible plants have size that may cause problem. Indeed, the Radish and Endive are small plants which all along their evolution will stay inside their hole. For the strawberries, it is different: the tree will grow big but not in the same growth module than the others. In fact, at a certain step of its evolution, the tree will continue its evolution in a module of type 3 and 4. Note that it is the only plant going through these types of modules. We will see later that the solution that we found is also valid for the strawberries.
So the three plants that will grow too much are Lettuce, Fennel and Cabbage. We found out that their diameter is under 25 centimeters in most cases. Again, this data is not very precise but it is the only that we can use.

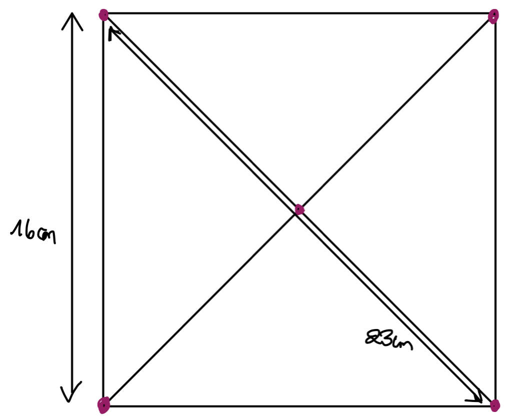


Figure 14: A growth module with its dimensions.

As you can see, the distance between a pair of nodes is at most 23 centimeters. We know call a 'big plant' a plant that can spread out from its hole. If we put a big plant in one corner, we need to have a small plant in the holes adjacent. Therefore if we want to plant other big plants, we can only plant another one and we must put it in the opposite hole of the diagonal. In other worlds, we need to have at most 2 big plants in one growth module. If we have 2 big plants, the only way that they don't touch themselves is if they are in the diagonals.

Therefore we add a constraint to avoid having more than two big plants in one growth module. When Section 5.1.4 when tackling the diversity problem, we already added a constraint: there must be at most 2 plants of the same species in a growth module. Since this constraint is applied for all species of plants, it is in particular applied for the the 'big plants'. We are left to implement the constraint for the pairs of different species.

Now, we know that we have to implement the constraint for the following 3 plants : Lettuce, Cabbage and Fennel. Let's denote :

- Lettuce := p_1
- Fennel := p_2
- Cabbage := p_3

We have two options. We either add the constraint for all growth module, which can be much easier to compute. Or, we add the constraint only for the modules type 2 which are the only ones where an issue can occur. We decided to opt for the second option since we want to minimize the loss of plants that occurs when implementing these constraints.

$$\sum_{d=1}^{size(k_p)} (x_{ijk_{p_1 d}} + x_{ijk_{p_2 d}}) \leq 2$$

$$\sum_{d=1}^{size(k_p)} (x_{ijk_{p_2 d}} + x_{ijk_{p_3 d}}) \leq 2$$

$$\sum_{d=1}^{size(k_p)} (x_{ijk_{p_1 d}} + x_{ijk_{p_3 d}}) \leq 2$$

where (i, j) corresponds to the edge in direction to a module of type 2.

Now let's get back to the case of the Strawberries. Since we have the constraint of diversity implemented for each growth modules. It is applied for the growth modules of type 3 and of type 4. We have that at most 2 trees of strawberries can be growing in these types of growth modules. Therefore there will be no problem with its size.

5.3 Analysis

We noticed that in our case this does not change our number of plants. At first, this seemed strange, so to verify, we ran the code for different inputs. With some hypothetical data, this constraint has indeed decreased the number of plants. Therefore, this constraint is in fact well implemented. And the non-diminution of the number of plants can come from the fact that the constraint of diversity already allows us to have less than two plants of the same type. Also, the number of days that a plant needs to grow is approximately the same for these three plants.

The running time is bigger now, it went from 3h30 to approximately 5h30. But, as said previously, this running time is still good in practice since the algorithm is meant to be used only once for the next 6 months.

6 Note On The Strawberries

GrowBotHub is considering 6 types of plants for its farm: lettuce, fennel, endive, cabbage, radish and strawberry. We noticed that the strawberries differs from all the other plants on many levels. First of all they are the only plants that don't stop their growth at a growth module of type 2, they stop at a growth module of type 4. On top of that, they take 120 days to grow when all the other plants take approximately 60 days to grow (except for the radishes which take 30 days). Finally each plant of strawberry produces 5 to 6 fruits as opposed to the other plants considered where each plant only gives one vegetable.

While testing the scheduling algorithm by changing the α variable of the balance constraint and the number of growth modules 3 and 4 (which are only used by the strawberries), we noticed that the total number of plants outputted was limited by the strawberries. Indeed, last semester, while implementing the balance constraint, they did not consider that each plant of strawberry can render more than one fruit. We need to update this constraint. To do so we tried two ways.

The first solution to overcome this issue was to remove the strawberries from the balance constraint and force them to grow every 10 days. That way the strawberries do not limit the number of plants for all other species in the balance constraint. When we tested this updated version of the code, our total number of plants went from 168 plants to 172 plants and did not change the running time. The downside of this method is that it is not very robust. If we consider less than 10 days then we would not get any strawberries, if we consider a larger `Horizon` than 180 days then we might get too many strawberries and less of other plants.

The second solution to surmount this problem is by multiplying each occurrence of strawberry by 5 (we consider for simplicity that each plant of strawberry gives exactly 5 fruits). When we tested this updated version of the code, our total number of plants went from 168 plants to 180 plants and it did not change the running time. Since there are no noticeable downside, we decided to opt for this solution.

7 Figuring Out The Optimal Number Of Modules

We were also informed during the semester that a new goal has been added and needs to be tackled. As a matter of fact, we thought that the number of growth modules was already known. But it turns out that it's not, we only know that it has to be somewhere between 12 and 16. Thus, our new task is to determine how many modules of each type we should have in order to produce the most optimal output. It may at first seem odd that, in order to produce the most optimal solution, we would not always take the maximum number of module (which is 16). In reality it might be necessary and optimal (since we have a balance constraint) to remove a module to let taller plant grow in the module under it.

The main idea is to run this new algorithm at first, get the most optimal combination of modules as output and gives this as inputs to the scheduling algorithm. To implement this new goal, we decided to do an exhaustive search. We will run the scheduling algorithm on every possible combination of number of modules per type of modules, and output the ones that produces the most plants.

Because we are going to try all possible association of modules, we want each iteration to run fast. Since each iteration consists of running the scheduling algorithm with specific inputs regarding the modules, we want to give to the code the old version of the scheduling algorithm where one node stands for a type of module. Even though we loose some information regarding the plants' size, we saw that this would not alter the total number of plants outputted (or alter it by a negligible number depending on the input) which is the only output of the scheduling algorithm we are considering here. Now each iteration of the code runs in a few minutes instead of a few hours.

8 Conclusion

To conclude this report, we are going to compare the algorithm before and after our project.

	Last semester	Now
Size of the graph	(14412 , 212'490)	(2892, 38'280)
Number of modules	8	16
Running time	MemoryError	5h30
Constraint Diversity	✗	✓
Plant's size	✓	✓
Number of plants	MemoryError	206

Table 1: Comparison of the scheduling algorithm before and after our project

As we can see in the array, the size of our graph is considerably smaller than before. Thus we allowed the code to run on real size data, which was not possible before. We also have a more accurate diversity constraint. In both cases, a constraint on the plants size was implemented, only theirs was based on hypothetical data and ours on real but vague ones. Finally, we were able to create an algorithm capable of finding the optimal combination of modules which is 7;7;1;1 for our inputs, whereas they were only able to test their algorithm with at most 8 growth modules since their problem size was too big.

In the end, we were able to return to GrowBotHub two algorithms. The new algorithm that determines the optimal combination of modules and an optimized version of the scheduling algorithm that has a good running time. The latter takes as input:

- the number of growth modules for each type of module (given by the new algorithm)
- the different types of plants and their properties (total growth time, number of days spent in each growth module, size of the plant over time. . .)
- the time over which we want to schedule (`Horizon`)

and outputs:

- a text file containing the instruction for each day for the robotic arm
- a text file containing the status of each growth module: for each module and for each day, it tells us what plants it contained
- a text file containing all the plants that have been harvested on each day, as well as, for all the species considered, all the day on which this specie has been harvested
- a text file containing the total number of plants as well as the total number of plant from each specie
- a png image of the graph given to the solver
- a png image of the graph outputted by the solver

The algorithm runs in approximately 5.5 hours while respecting the constraints given by GrowBotHub. In the end, we can affirm that in 6 months, our algorithm will output the instruction that will allow to harvest 206 plants (note that each plant of strawberry produces more than one fruits).

8.1 Next Year's Project

Let's define the problem of rescheduling: it consists of an algorithm that computes on the spot a new schedule if an unexpected event occurs. This is determined by a robot that measure the volume of a plant on day d in a growth module with the help of a camera and compares it to a given map. If the measured volume is smaller the expected volume of this specie on day d, then this plant needs more time to grow in which case we need to compute a new schedule. Unfortunately, we didn't have time nor the data on the plant's size to approach this problem. This task is the new goal of next semester scheduling project.

9 Annex

9.1 The inputs

- 4 types of growth modules
  - 7 trays of module 1 (”seedling”)
  - 7 trays of module 2 (”vegetative growth”)
  - 1 trays of module 3 (”flowering”)
  - 1 trays of module 4 (”development”)
- `Horizon`: 180 (6 months)
- 6 types of plants
  - Lettuce: 30 days in module 1, 25 days in module 2
  - Endive: 20 days in module 1, 40 days in module 2
  - Cabbage: 20 days in module 1, 30 days in module 2
  - Fennel: 20 days in module 1, 30 days in module 2
  - Radish: 15 days in module 1, 15 days in module 2
  - Strawberry: 49 days in module 1, 21 days in module 2, 28 days in module 3, 19 days in module 4

9.2 The graph

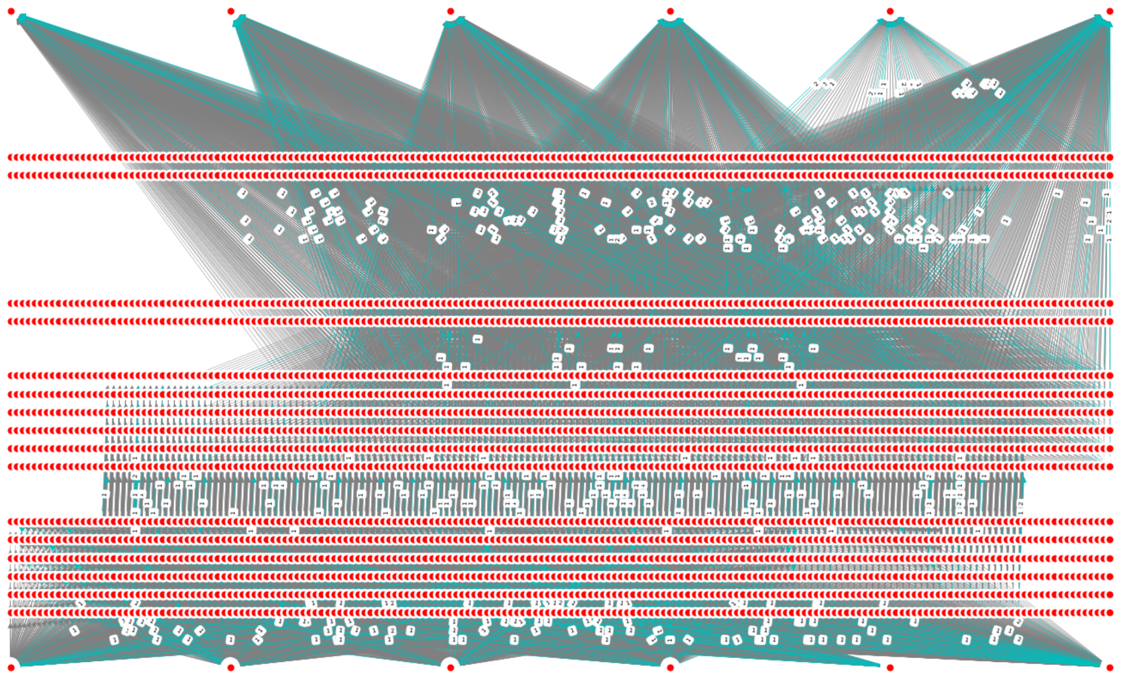


Figure 15: Graph outputted by the scheduling algorithm. The blue edges are the one used by the solution

9.3 Extract of the ”instruction” file

```
...
DAY 61
Move 1 Lettuce from module 2 of type 1 to tray module 3 of type 2
Move 1 Radish from module 5 of type 1 to tray module 1 of type 2
Move 1 Endive from module 4 of type 1 to tray module 3 of type 2
Harvest 1 plant Endive from module 3 of type 2
Plant 1 seed of Fennel in module 5 of type 1
DAY 62
...
```

9.4 Extract of the ”status of a module” file

```
GROWTH MODULE TYPE 0
DAY 0
Module : 0 — Plant : Fennel
Module : 0 — Plant : Radish
Module : 1 — Plant : Cabbage
Module : 1 — Plant : Fennel
Module : 1 — Plant : Radish
Module : 2 — Plant : Endive
Module : 2 — Plant : Cabbage
Module : 3 — Plant : Endive
Module : 3 — Plant : Radish
Module : 4 — Plant : Radish
module : 4 — Plant : Endive
Module : 5 — Plant : Radish
DAY 1
...
```

9.5 Extract of the "harvesting data" file

...
DAY 151
Harvest 1 plant Cabbage from from module 3 of type 2
Harvest 1 plant Cabbage from from module 2 of type 2
Harvest 1 plant Cabbage from from module 5 of type 2
DAY 152
Harvest 1 plant Strawberry from from module 3 of type 4
...

Endive:
Day 61: Harvest 1 plant Endive from module 3 of type 2
Day 62: Harvest 1 plant Endive from module 4 of type 2
Day 63: Harvest 1 plant Endive from module 1 of type 2
Day 64: Harvest 1 plant Endive from module 2 of type 2
...

Cabbage:
...

...

Lettuce	Endive	Cabbage	Fennel	Strawberry	Radish	
...	...	...	...	...	...	...
80	0	1	0	0	0	0
81	0	0	0	1	0	0
82	0	1	0	0	0	0
83	0	1	0	0	0	0
84	0	1	0	0	0	0
85	0	0	0	0	0	1
...	...	...	...	...	...	...