École Polytechnique Fédérale de Lausanne

Scheduling of GrowBotHub

by Nicolas Vial and Augustin Henry

Autumn 2020

Bachelor Thesis

Prof. Friedrich Eisenbrand
Thesis Advisor

Jonas Racine
Thesis Supervisor

Acknowledgments

We want to say thank you to our assistant Jonas Racine who has helped us along the way. He gave us really useful advice and we learned a lot this semester with him. Also a big thanks to Aurélien Debbas, the president of GrowBotHub, and all the members of the team. It was really nice to work with you. Every time we had a question, you quickly helped us with lots of positivity, this has permit us to work in a serene environment.

Nicolas Vial and Augustin Henry

Contents

Chapter 1

Introduction

1.1 GrowBotHub

This report concludes our semester project, which consists of making a program for the scheduling of GrowBotHub. GrowBotHub is a association of EPFL. The objective is to grow vegetables in hostile conditions like the ones in space or on the moon in an automated way. The plants will grow in holes with an aeroponics technique. This consists in feeding the roots of vegetables by sprinkling water containing appropriate nutrients. There will be 5 holes per Growth modules that are disposed on shelves. This project is implemented jointly by multiple sections of the EPFL and has different sections :

- Structure : design the shelves, the growth modules and displacement of the robot.
- Vision and Scheduling : the section in which we are working, designs the algorithm for the movements of vegetables across the different growth modules and uses some data vision to verify the growth of vegetables before moving or harvesting them.
- Aeroponics : the project uses hydroponics and aeroponics techniques to grow the plants. They plan the nutrients to be used when watering the plants.
- Robotics : they will code all the programs for the different movements of the robotic arm that will move the plants.
- Networking : they design all the communication systems between the different components of the project and a remote control from earth.
- Chemical-engineering : they are closely working with the aeroponics team to design the different nutrients and the use of waste.

1.2 The scheduling goal

The goal of the scheduling section is to develop an algorithm capable of optimising the number of plants produced and give the different changes of position that the robotic arm will have to carry out.

Since everything is closely correlated to the bigger project, we have to comply to the different constraints of GrowBotHub, we have specific inputs and we have to deliver agreed outputs.

We are being given multiple inputs :

- Number of growth modules and number of holes per growth module.
- Different types of growth modules.
- Sequences of growth modules (or trays) for each species of plant. Since we are using aeroponics, the plants have to move between different growth modules to get the correct nutrients at the appropriate time.
- Distances between holes.
- Optimisation horizon, which is the period that we should schedule, usually 6 months.
- Unit of time, ideally 1 day : we would move and harvest plants once a day.

We agreed that our program must output two items. One is the list of instructions that the robot should do everyday, it is given in day per day list :

- Day1
    1. Hole 1, tray 1 $\rightarrow$ hole 3 tray 2
    2. Hole 4, tray 2 $\rightarrow$ hole 1 tray 1
    3. Etc...
- Day2
    1. Etc..

The second one is the status of each growth module on each day. We indicate for each hole if there is a plant and its name.

Our optimisation must comply with certain constraints given by GrowBotHub :

- Maximum one plant at a time in each hole.
- The sequences of growth modules for each species of plants must be applied.
- Distance between holes given in input must be respected to prevent the plants from overlapping each other.

Chapter 2

Design

2.1 First approach

We decided to represent the problem with a graph. The nodes would stand for holes and the edges for possible movements of the plants between holes. One question quickly rose: how would we represent the time in our graph? The algorithm must optimise the scheduling on a given time window and we should verify that the plants stay long enough in each growth module. We solved this problem by using copies of the nodes, we make one copy for each unit of time; it is a time expanded graph.

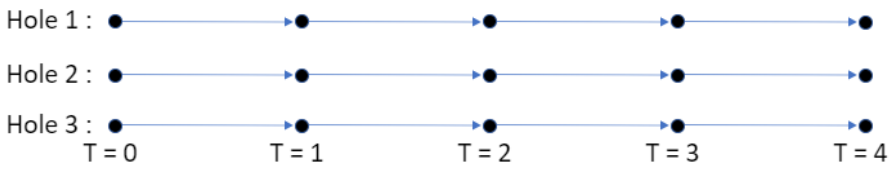

Figure 2.1: Time expanded graph.

For our modelling to work, we must force the plants on a certain path through the graph. A plant has to be seed on a certain day, stay a given time in each growth module following its specific sequence and then be harvested. The path of each plant is decided by the edges. The first idea is to connect the nodes in a certain way to form one path to be followed by the plant. There is an issue with that straight forward solution. We want to have the possibility to put seeds every day in the system and there are multiple species of plants to consider. We can't create one unique path for each possibility of plants, the paths would cross over each other. So we must create all the possible connections in our graph and put conditions on the edges for each plant. The path would be forced by the conditions. This easily works for the different species of plants, but for the same plant we must consider all the possible entry time. We decided to consider one commodity per plant per unit of time. Each commodity is an object in itself, and each edge in the graph will have particular conditions for each commodity.

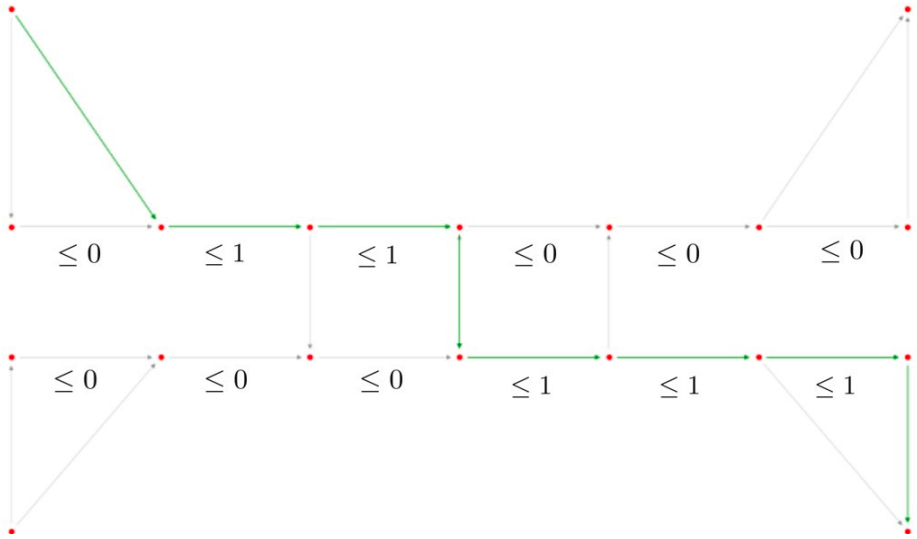

Figure 2.2: One commodity and the conditions on the edges

We need one variable per edge per commodity to represent the flow.

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

Where i and j are the starting and ending nodes of the edge, k the commodity, E the set of edges and K the set of commodities.

We have one source and one sink per species of plants. Each source creates one commodity per unit of time and connect an edge to the graph until there is not time for the plant to grow. The sources receive the possible plants; they have one edge per commodity.

2.2 Optimisation

Now that we have represented our problem with a graph, we have to decide how to optimize it. The aim is to maximize the number of plants produced. It is the same as maximizing the sum of inflow to the different sinks.

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

With S the set of sinks, x is the variable we just talked about, it represents one commodity on one edge. The δ^+ is the set of edges entering the node, in this case, a sink.

With this formula as goal, we have to run a max-flow algorithm on the graph. We are using a solver for this part, first we used Gurobi then we switched to Pulp afterward, we will discuss why later on. The solver needs constraints and an objective to solve the problem. The objective is to maximize the entering flow into the sinks, but we are missing some constraints. For a graph to

be correctly described we have to implement the conservation of the flow. Without it, the solver would simply add commodities of plants on the edges to the sinks, ignoring the others.

$$\sum_{(i,j)\in\delta^+(n)} x_{ijk} - \sum_{(i,j)\in\delta^-(n)} x_{ijk} = 0$$

The sum of flow entering all nodes (δ^+) except for the sources or the sinks must be equal to the flow exiting from them (δ^-).

With those tree constraints : the solver will use all the edges, each path for the plants will be used. At this point the max-flow is simply a full graph, we are missing the constraints given by GrowBotHub that add rules between the plants.

2.3 Constraints

As said before, there must be at maximum one plant per hole at any time. We can implement it by constraining the sum of all commodities on every edge to maximum 1.

$$\forall i,j \in E \quad \sum_k x_{ijk} \leq 1$$

With our actual implementation, we have already a good approximation of our final result. By looking closely to the optimisation result, we saw that there were some issues to rectify. For example, the optimisation at this state will privilege some of the species of plant, one factor is the time plants take to grow. Some species get not involved in the result. This is why we have some side objectives that we will consider later on.

Another given constraint by GrowBotHub is to be careful that the plants do not overlap on each other, for that we have the distances between holes and the pairs of neighbor holes. The solution we chose was to sum up all the current size of the commodities entering the two neighbor holes and constraint it to be at most the distance between those holes. This will make sure the plants do not overlap each other.

$$\sum_{(i,j)\in\delta^+(n)} x_{ijk}.growthRate_{ijk} + \sum_{(i,j)\in\delta^+(h)} x_{ijk}.growthRate_{ijk} \leq sizeMax_{(n,h)}$$

The two nodes n and h are neighbor holes. We get the current size by multiplying the size the commodity k would have at the edge (i,j) that we call the 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.

2.4 Secondary objectives

The main objective of our optimisation for the scheduling is to produce a maximum of vegetables. We have some unbalances between the different species of plant; some of them grow longer than others, some are bigger than others. Those unbalances make the max-flow tend to put aside some species that are less advantageous for the number of plants produced. The aim of GrowBotHub is of course to maximize the number of plants, but also to have a diversity, they want to grow each species of plants more or less equivalently.

This is why we implement some balancing constraint for our graph, the aim is to force the max-flow algorithm to grow every species even if it reduces a bit the main objective. For that we forced the number of plant produced from each species to be close to the average, which is the total of plants produced divided by the number of different species.

$$\forall s \in S \sum_{(i,j)\in\delta^+(s)} x_{ijk} \geq \frac{\sum_{s}^{S} \sum_{(i,j)\in\delta^+(s)} x_{ijk}}{N} - \alpha \quad , \alpha \geq 0$$

With N the number of different species and α a manually set constant that is used to loose up the constraint. It is chosen in function of the aim of the algorithm, smaller it is, the more balancing the result will be, but it might reduce the total number of plants.

Another side objective we had to implement is the homogeneity of productions over time, the ultimate goal is to be able to harvest plant everyday. We had different ideas to implement it but finally we decided not to do it. Actually, our current optimisation already has a very good homogeneity, due to the fact that the best solution is to divide the production of plants over time and the balancing also participates to the homogeneity. Also our solutions were very costly and would add lot's of computation thus solving time for of the solver, as we will discuss later, our solving time is already bad so it would not be worth it.

Chapter 3

Time analysis

3.1 First tests

During the period of creating the design of our algorithm, we performed multiple tests that allowed us to verify and correct our implementations. But, in order to be able to verify by hand that the results were correct, our tests were all very small and unrealistic compared to the real quantities. So once the first draft design was finished, we started to perform tests with realistic values, and therefore much larger. We thus carried out our first major tests with Gurobi with a graph having the shape of the toy example described in figure 3.1 but with much more plant species and a larger optimization horizon.

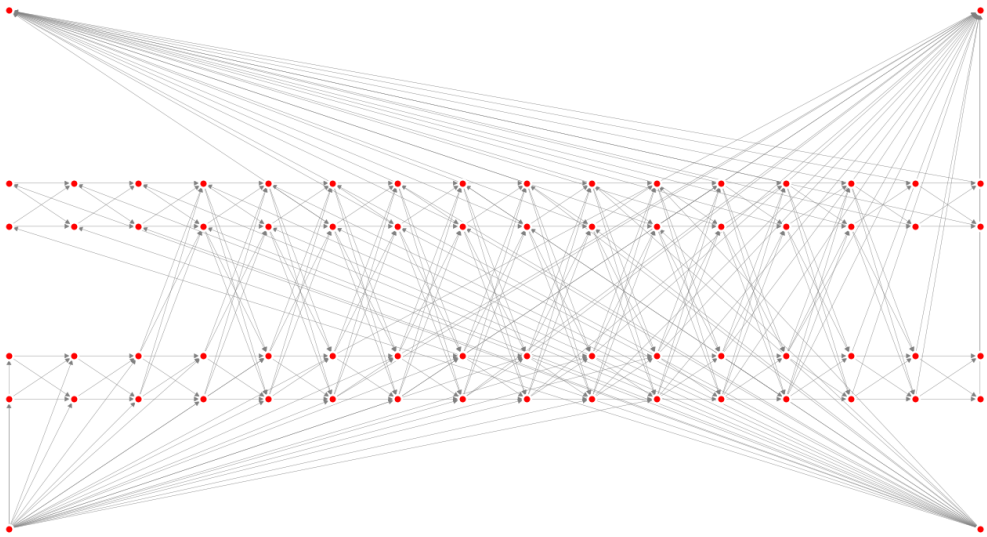

Figure 3.1: Toy example graph containing two growth modules with two holes each and an optimization horizon of 15 days.

The first thing we noticed while performing realistic tests (but with an optimization horizon

smaller than the realistic one) was that Gurobi was no longer able to give us an optimal solution in an acceptable time (at least more than 4 hours). So we found a simple solution to this problem: forcing Gurobi to stop after a certain amount of time, called timeout. This choice is obviously not without sacrifices because as Gurobi did not have the time to find the optimal solution, we lost a certain number of plants. But after several tests and measurements, we noticed that Gurobi found a solution very close to the optimal one quite quickly, so we lost a small number of plants as shown in Figure 3.2.

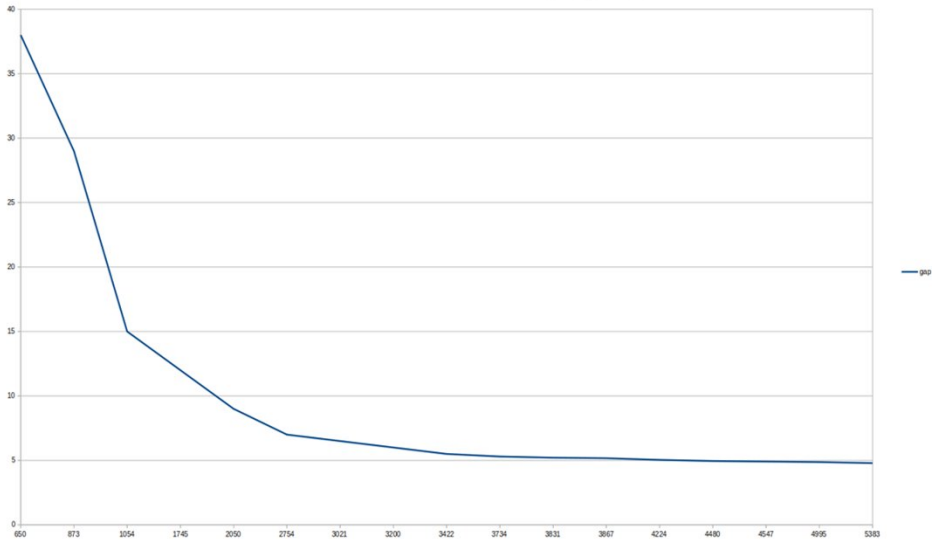

Figure 3.2: gap with the optimal solution in function of time where the gap is the percentage between the solution found and the optimal solution.

3.2 Fully realistic tests problems

Now that we could get a result thanks to the timeout, it was time to perform the tests with realistic values. We encountered a major problem very quickly, the solver didn't have enough memory to store all the values for all the edges of the graph.This problem made us raise a major question to which we didn't have the answer: Are diagonal edges in each growth module useful? To find out, we performed tests as shown in figure 3.3.

max size	with diag	without
8	28	28
9	203	196
10	224	210
11	238	238
12	245	245
13	245	245
	time out	

Figure 3.3: Number of plants produced in function of the maximum size, with and without the diagonal edges. Orange results have been reached after a timeout.

We clearly see that with a relatively strict maximum size, we get more plants with diagonal edges. For the continuation of our project, we decided to continue without the diagonal edges in order to obtain results but we left the choice to put them in the code because we know that this project will be further improved which will allow to put back the diagonal edges and thus to produce a maximum of plants. We therefore ended this project with a graph in the form of the one shown in Figure 3.4.

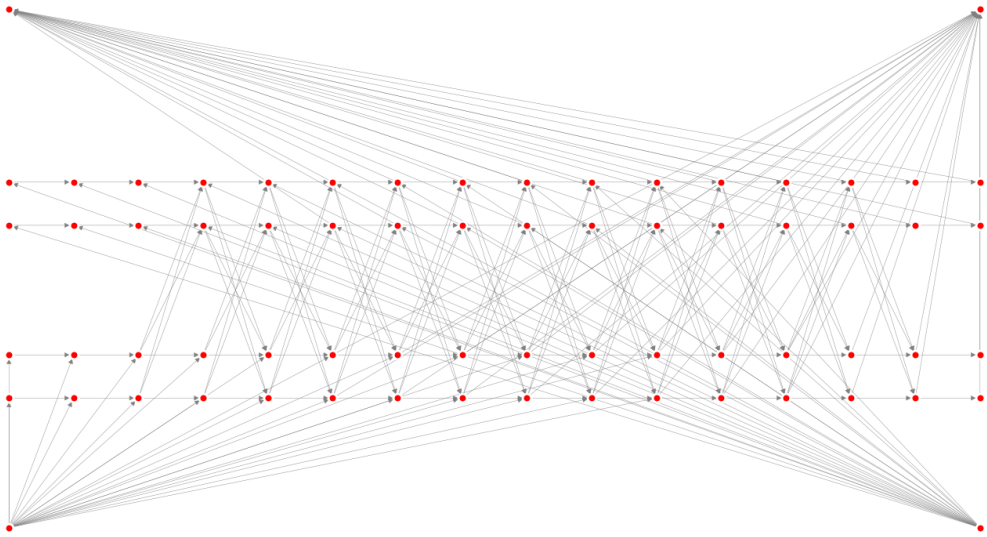

Figure 3.4: Same graph as in the figure 3.1 but without the diagonal edges.

3.3 Final time analysis

Finally, we had to change our solver from Gurobi to Pulp because GrowBotHub requires to use open source programs. Unfortunately, Pulp is three to four times slower than Gurobi, so if we keep the same timeout between the two solvers, the result with Pulp will always be smaller or at best equal to the result obtained with Pulp. In spite of this, we must keep in mind that our algorithm is only slightly optimized to allow the solver to find an optimal result as quickly as possible. This project will therefore be resumed and improved in part to solve this problem of solving time.

Chapter 4

Conclusion

4.1 The program

We have to give GrowBotHub a working program they can insert with the other sections' work. We are writing our code in python, it is highly modular so it can be easily modified or improved. For the solver we had to use an open source solver so we switched from Gurobi to Pulp. This add another difficulty, Pulp is three to four times slower than Gurobi, this is a real issue when we are working with big inputs.
The pipeline is pretty straight forward, we first read the given inputs, then we construct the graph, we also draw it for a better comprehension and debugging, with the graph we use the solver to optimise the production and find a solution, we draw the optimised graph and we output the result in two different text files. The drawing of graphs is optional as it is not essential and takes some time.

4.2 Next project

Our project will be continued next semester to be improved. The next project will use what we have done so far and add some features. The main objectives are improving the solving time, maybe modifying some parts of our approach for a better one, this will allow us to optimise the scheduling on bigger horizon with a better precision. Also they will add a possible initial state, run a scheduling on growth modules with already some plants in it. Our section in GrowBotHub is also working on data vision to recognise the vegetables, we want to use it to verify the good timing for moving the plants or harvesting them. This will make our scheduling more flexible for the plants' growth.

Chapter 5

Appendix

5.1 Project's location

All the code is grouped on a repository where all the use and the code are explained : `https://github.com/GrowbotHub/Scheduling`

5.2 Input and outputs

Here is an example of possible inputs (figure 5.1) :

```
TRAYS|8
HOLES|5
HORIZON|15
MAX_SIZE|15
MAX_TIME|20

PLANT|Lettuce|5|b|1,2,3,4,5|0,1|0,2;2,5

PLANT|Fennel|5|g|1,1,3,4,6|1,0|0,3;3,5

PLANT|Strawberry|5|y|1,1,3,3,5|6,4,1|0,1;1,2;2,5

PLANT|Endive|5|purple|1,2,4,6,9|0,2,7|0,1;1,4;4,5

PLANT|Cabbage|5|orange|1,2,4,6,8|6,7,5|0,1;1,3;3,5

PLANT|Raddish|5|c|1,2,5,6,7|4,7|0,4;4,5
```

Figure 5.1: Inputs example. For more details, see the README on the repository linked above.

We then have two types of outputs, the first one (figure 5.2) gives all the moves that the robotic arm has to carry out over time and the second output (figure 5.3) gives the states of the growth

modules over time.

```
DAY 2
Move the Endive from hole 1 and tray 0 to hole 1 and tray 2
Move the Endive from hole 2 and tray 0 to hole 2 and tray 2
Move the Cabbage from hole 2 and tray 6 to hole 3 and tray 7
Move the Cabbage from hole 3 and tray 6 to hole 1 and tray 7
Move the Cabbage from hole 4 and tray 6 to hole 2 and tray 7
Plant a seed of Endive to hole 0 and tray 0
Plant a seed of Endive to hole 2 and tray 0
Plant a seed of Endive to hole 4 and tray 0
Plant a seed of Cabbage to hole 0 and tray 6
Plant a seed of Cabbage to hole 3 and tray 6
Plant a seed of Cabbage to hole 4 and tray 6
DAY 3
Move the Cabbage from hole 0 and tray 6 to hole 0 and tray 7
Move the Cabbage from hole 4 and tray 6 to hole 4 and tray 7
Plant a seed of Fennel to hole 1 and tray 1
Plant a seed of Fennel to hole 2 and tray 1
Plant a seed of Fennel to hole 4 and tray 1
Plant a seed of Endive to hole 2 and tray 0
DAY 4
Move the Endive from hole 0 and tray 0 to hole 0 and tray 2
```

Figure 5.2: first output example. For more details, see the README on the repository linked above.

```
GROWTH MODULE 0
DAY 0
Hole : 1 | Plant : Endive
Hole : 2 | Plant : Endive
DAY 1
Hole : 1 | Plant : Endive
Hole : 2 | Plant : Endive
DAY 2
Hole : 1 | Plant : Lettuce
Hole : 3 | Plant : Lettuce
DAY 3
Hole : 0 | Plant : Endive
Hole : 1 | Plant : Lettuce
Hole : 2 | Plant : Endive
Hole : 3 | Plant : Lettuce
Hole : 4 | Plant : Endive
DAY 4
Hole : 2 | Plant : Endive
Hole : 3 | Plant : Fennel
Hole : 0 | Plant : Fennel
Hole : 1 | Plant : Fennel
Hole : 4 | Plant : Fennel
```

Figure 5.3: second output example. For more details, see the README on the repository linked above.