Modeling and Simulation

Cooking Process Simulation

Klaus Derks, 01529340 - 066 926*
Dávid Lukács, 12306344 - Curriculum Number†
Martina Mlezivová, 12306360 - 033 266‡
Lana Ramšak, 12302817 - 033 201§

February 6, 2024
Supervisor: Martin Bicher

Abstract

Abstract comes here ...

*....
†...
‡...
§...

Contents

1 Introduction

In the past years the industry put an emphasise on being the fastest and most efficient. Take-away is getting more and more popular. Next day delivery is not as rare as it used to be and products are rarely hand made. Optimizing the process plays a big role in that. A simple example can be cooking, where the order of tasks significantly impacts the efficiency. When following a recipe, you have to think in advance what you will need, and how much time it will take. If you don't turn on the oven before starting, you might waste time waiting for it to heat up later. In our project we will be trying to find an optimal ordering of tasks, to make the cooking process more efficient. This way recipes can be done quicker and the work can be better distributed.

For our starter recipe simulation we choose tried to choose something basic. The recipe that came to mind was the dish Goulash. Here is the following recipe:
(myb add recipe)

As you can see it includes a lot of different ingredients and cutting tasks, which are fairly simple to include in our simulation but help set the start model. Furthermore it is a recipe where the order of the tasks is of high importance, which is something you have to be aware of when cooking.

For our second recipe we chose the dish named Potatoe salad. We made our choice cause on this dish we can show the importance of the task order and the best time management. This dish includes a lot of tasks of cutting and boiling.

The recipe of Potatoe salad is following:

2 Model Implementation

Our program is written in the programming language Python. We made that decision because that's the program we are all familiar with and are able to find our way around.

2.1 Data storage

Tasks are implemented as objects, where each one has it's own

- `name`, stating the name of the task,
- `duration`, meaning the time it takes for the task,
- `required task-prerequisites`, which is a list of tasks-names that have to be completed before,
- `required resources`, meaning all items needed for the task for example *cooking stove, knife, cook*

Our recipes are portrayed as lists of tasks amd saved in a separate `json file`. Our kitchen resources are also defined in another `json file`, where each one is assigned a `name` and `num`, meaning quantity.
A few parameters are also set at the start of the code. One of these is the name of the recipe file and the difference of time factor, which we will talk more about later. To save the date changing throughout the cooking process we constricted a class cooking, where we define these properties:

- list of completed tasks, which is at first empty, later on we will add the completed tasks here,
- list of resources, imported from the json file,
- list of tasks, also imported from the json file,
- copy of list of tasks, that won't change throughout the simulation
- order, where we save the order of tasks in current simulation

2.2 Recipe Simulation

For the actual simulation, meaning for a recipe to be excecuted, we defined another class named DiscreteEventSimulator. Everytime we construct a new one, we will also construct a new class cooking. This way all our parameters are everytime set back to the start (ex. list of completed tasks). The class also has a parameter `event_queue`, where scheduled events are being added, and parameter `time`.

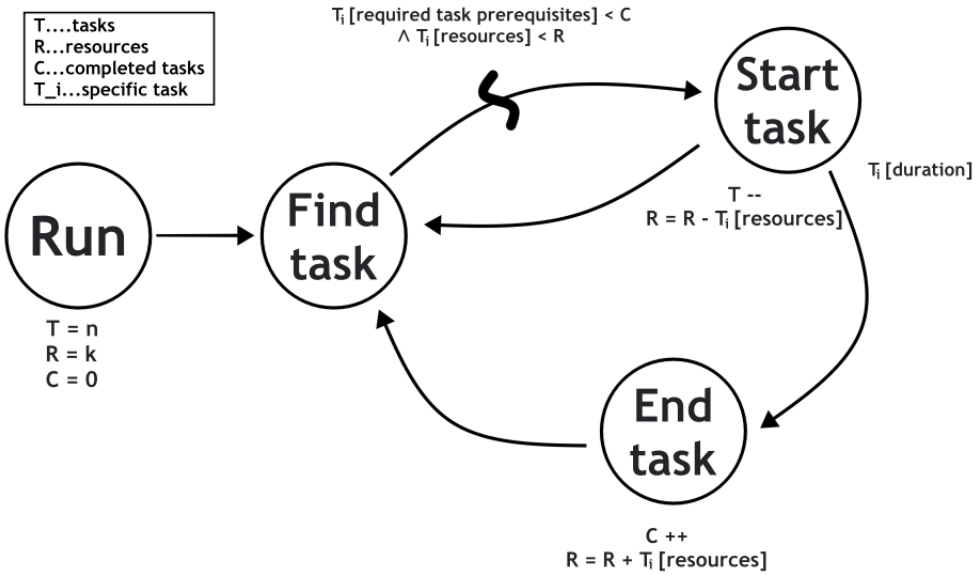

Figure 1: Discrete simulation graph of a cooking process.

Here we can see a simplified graph on how the simulation works. We start by calling the method `find_task`, this as suggested searches for a task that can be excecuted, meaning all resources needed are available and all tasks that needed to be done before are already completed. If we find such a task, method `start_task` is called, where the previously mentioned resources are taken of of the list, the task name is added to the order and the method `schedule_end_task` is invoked. There we make an event, where we calculate the time it will be finished, by adding up duration to 'current time' and save the name of the task. This is appended to the `event_queue` list. Right after the method `find_task` is executed again.

Right at the start we start a while loop, that will be done once the number of completed tasks is the same as the number of tasks at the start. With each iteration of the loop, the time increases by one. We also use this to check if there is an event scheduled at the current time. In case there is one we call the method `end_task`. This is where the previously removed resources are added back to the list and the task is added to the list of completed tasks.

At the end the function `run_simulation` returns a touple, where the fisrt component is the time it took, and the second the order in which the tasks were completed. Now that we got our program to execute a recipe, we started to work on finding the optimal order of tasks, to get the shortest time.

2.3 Different orders of tasks

The first idea we got, was just to try all possible permutations of the list of tasks, run the simulation for each one of them and then choose the one with the shortest time. To help with understanding the code and making sure it worked well, we made sure to get a list of all possible orders and the time it took. Furthermore due to prior task prerequisites the recipe we had at the start returned us the same duration, no mater the order of tasks. To fix it we added a task, to have a three-task process, meaning heating the water, peeling potatoes and cooking them. Moreover, we changed some of the durations of tasks.

When running the code we realized, the order of the last few tasks rarely changes. This

was because or the prior task prerequisites, since these tasks can only be once the others are completed. Here you can see some of the possible orders we got.

```
[cutting meat, heating water, cutting onions, cooking meat, frying onions, cooking potatoes, cooking everything]
[cutting meat, cutting onions, heating water, cooking meat, frying onions, cooking potatoes, cooking everything]
[heating water, cutting meat, cutting onions, cooking meat, frying onions, cooking potatoes, cooking everything]
[heating water, cutting onions, cutting meat, cooking meat, frying onions, cooking potatoes, cooking everything]
[cutting onions, cutting meat, heating water, cooking meat, frying onions, cooking potatoes, cooking everything]
[cutting onions, heating water, cutting meat, cooking meat, frying onions, cooking potatoes, cooking everything]
```

This gave us the idea, that instead of testing all possible orders, we should firstly check which tasks are required to be done first and put those at the start of the list. Furthermore, the ones who are required for more tasks, should be done first, meaning put at the start of the list, and others which are not required for any task, should be left behind. With this in mind we wrote the function `smart_permutations`, which takes the name of the recipe json file and returns some permutations of the tasks, where we know one of them will give us the best order. Here we can see the simplified process:

```
- get tasks from json file
- for task in all tasks:
    get task prerequisites (can be repeated)
- count repetitions of tasks
- sort in lists based on number of repetitions
- get all other tasks
- permutate each task list separately
- make all possible combinations
```

This is then inputed into the function `run`, which tries all the given orders and returns the one with the shortest duration.

2.4 Introducing randomness

Since cooking doesn't always go as planned, we added some randomness, by prolonging/shortening the duration of tasks. We started by defining a parameter `time_change_factor` at the start of the code, which states for maximum how many 'seconds' can a task be prolonged or shortened. This is all done in the function `time_randomness` that is called when scheduling a task and calculating the time it will be completed. The following graph of our simulation is then this, where the function f represents the previously mentioned function.

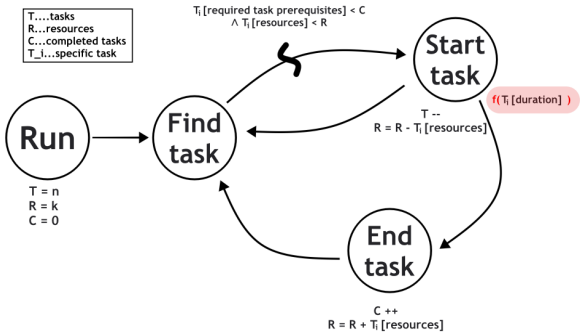

Figure 2: Discrete simulation graph of a cooking process with time randomness.

3 Simulation Results

For the simulation results we looked into a few different things.

3.1 Effect of different ordering

We wanted to see how big of an impact has the order of the tasks. That is why we looked for the best order for the recipe for goulash, and the worst one. This means the one that is done the quickest and the one that takes the longest. The following results are done with two cooks and time randomness.

```
BEST ORDER: 168,
[cutting meat, peeling potatoes, heating water, cutting onions, cooking meat,
cooking potatoes, frying onions, cooking everything]

WORST ORDER: 223,
[cutting onions, cutting meat, heating water, peeling potatoes, cooking meat,
 frying onions, cooking potatoes, cooking everything]
```

We can see that the difference in duration is 55 units of time, which is not small. This happens due to probalistic duration of time, but even without that, the difference would still be 30 units. The reason for that is the order of the first few tasks, which have to be done before cooking potatoes and cooking everything. If they are not done in the best order, the simulation has to wait for a task to be finished before doing anything new.

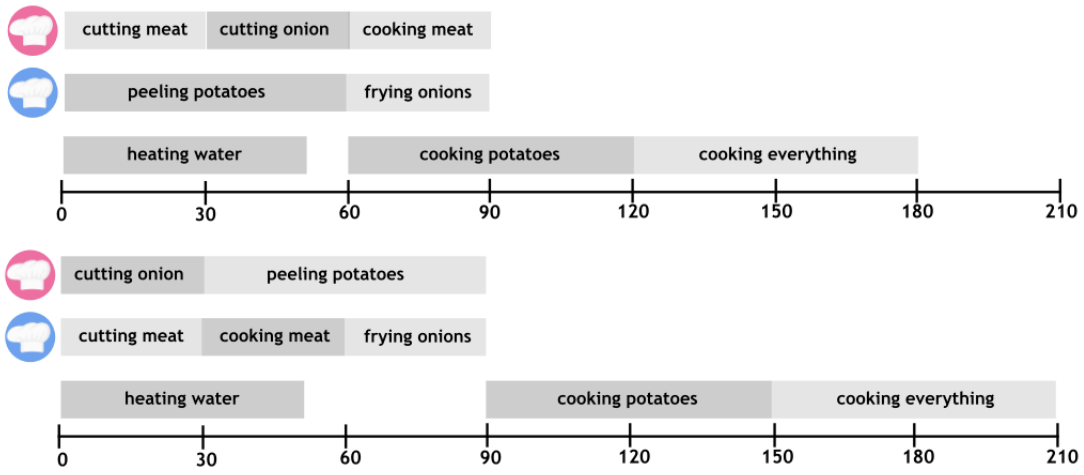

Figure 3: The best and worst order of tasks.

After seeing the difference an order of tasks can have, we were interested to see what are all possible durations of the recipe and which are the most common. In the histogram below we can make out that most of the time the recipe is completed in between 179 to 184 units of time, this suggests the order was similar to the best order as seen above, with just some different permutations of the first few tasks.

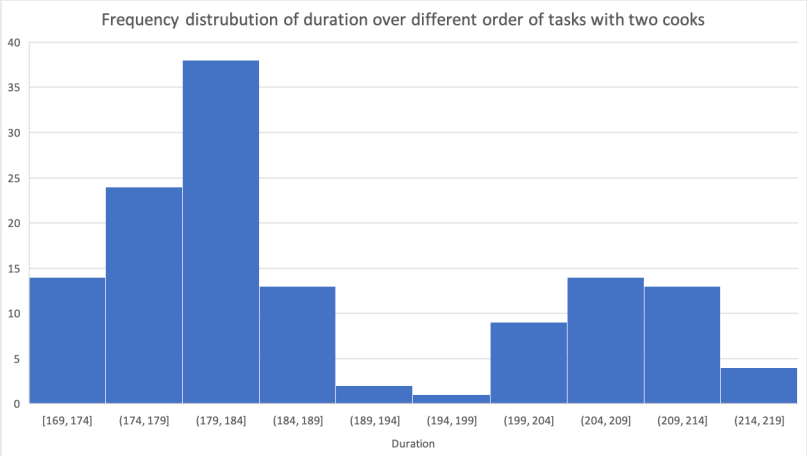

Figure 4: All possible durations of the recipe with two cooks, devided into groups by time interval 5 units.

We also did the same calculations for just one cook, but got a somewhat different distrubution, with most of the odrers gathered in the middle. In this case the different durations were only a result of probalistic time assignment, because when we ran the stohastic model, we only got one option of duration which is 251. This can be seen on the graph where most orders are between 243 to 248 units of time.

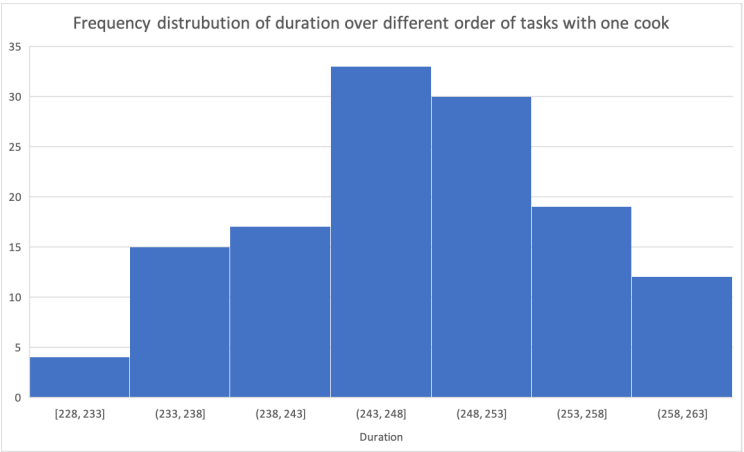

Figure 5: All possible durations of the recipe with one cook, devided into groups by time interval 5 units.

3.2 Recipe comparison to cook book

For our second dish we decided to do potato salad. We found a recipe in a cook book and adjusted it to our program. We wanted to test if the calculated optimal order matches the order in the cook book. Here we can see the two orders:

```
COOK BOOK:
- cooking potatoes
- cooking eggs
- cutting pickles
- cooling eggs
- cutting onions
- peeling eggs
- cutting eggs
- peeling potatoes
- cuttting potatoes
- make dressing
- mixing enerything
```

```
OPTIMAL ORDER:
- cooking potatoes
- cooking eggs
- cutting onions
- make dressing
- cutting pickles
- cooling eggs
- peeling eggs
- peeling potatoes
- cutting potatoes
- cutting eggs
- mixing enerything
```

We can see that the two orders don not differ much. Some tasks are done in different order, but those are mainly interchangable. One different thing is when the dressing is done. But this is because in real life you would want it to be freshly done before completing the salad, and our program does not take this into account.

3.3 Different number of resources

Lastly we focused on the number of resources to see if it really has that big of an effect on duration. We ran the program 5 times for each number of cooks and took the average of the optimal time. We gave the other resources a high enough quantity so that it won't effect the duration. These are the graphs we got.

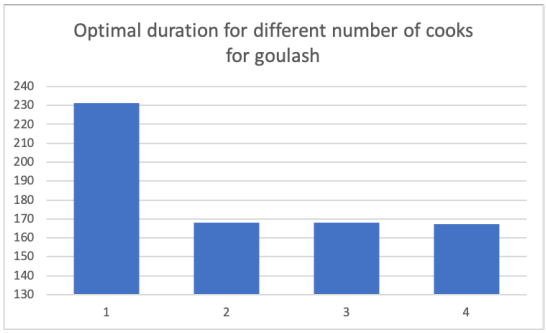

Figure 6: Duration of recipe goulash with different number of cooks

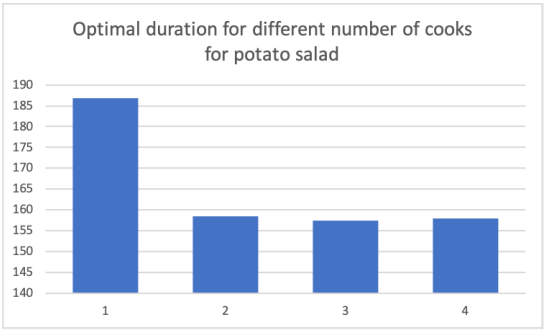

Figure 7: Duration of recipe potato salad with different number of cooks.

As you can see there is only a difference when using one or two cooks, after that you can use as many as you want, but the duration won't change. This is because the tasks in our recipes already have an order in which they have to be done and even if you have another cook, he still has to wait for the other task to finish. To test our program we decided to make another recipe, where we tried to do it so, that a different number of cooks will have a bigger effect.

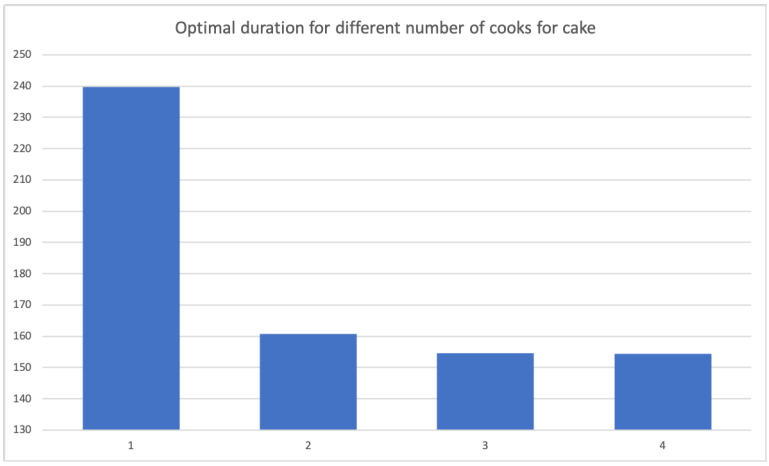

Figure 8: Duration of recipe cake with different number of cooks.

These are the results we got with implementing the recipe of baking a cake. We wanted to have more independant tasks and less strict prior ordering. In this recipe you have to do a lot of tasks independantly and then there is just a few tasks that combine all of them. This way you can see a difference between one, two and three cooks, but when it comes to four, the duration still remains the same. FUrthermore the difference was very small, only 6 time uhnits. This brings us to the conslusion that with every recipe there will be a point where the number of cooks doesn't matter anymore.

Besides observing the effect of the number of cooks, we did the same for other resources. Here you can see the duration for making goulash when having:

- double of everything: 181
- only one pan: 191
- only one knife: 181

This is because in the optimal recipe two knives are never used at the same time, while the pan is used for cooking the meat and frying the onions which is done at the same time.

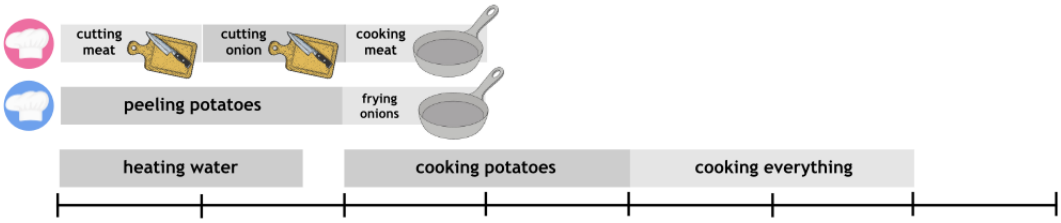

Figure 9: Use of pan and knife in the optimal order of the recipe goulash.

4 Conclusion

In conclusion, we breathly summarize our group work. Theme of the work was Cooking Process Simulation. We chose the program language Python. As was said before, we see some benefits in using it. also to use the json files to be the program more user friendly and also to have it more visually clear.

The results, that have been gotten, are describe in detail in the previous chapter. The main goal is ....

We conclude that We enjoyed the group work. We are familiar that our simulation program can get improved in the future. Unfortunately, there are some missing points-small mistakes. However, we hope that they have no that big impact on the running the program and the simulation in general.

- problems code, did in simpy, turned ou to not be good
- three recipes, simulate the results, some especially with a cause
- the order does affect the time, best to do long tasks first, with more cooks there is a diffrence, with one it is almost not important
- got similar order, didn't take into account things like being fresh, with some tasks order not important
- number of cooks only important to some number, big difference between 1 and 2, resources highly effect the time, need to have more pans/knives not just cooks

References