SDSS - Seats distancing software support

Fabio Vaccaro

29 Dicembre 2020

Contents

1 Introduction

During the Covid-19 pandemic period social distancing became a necessity. People have to keep a security distance from each other to avoid any possible virus transmission. There was initially a very heated debate regarding the right distance to ensure security. After a while, maybe for practical necessities, security distance was set to one meter. Restaurants and any public place where tables and seats are present have to respect those security precautions: seats and tables too close to each other should be disabled and as a consequence commercial establishments' capacity drastically decreased.

In such a situation became clear the need for a software to support restaurant owners with some kind of software to exploit their space in the best possible way. For this reason I thought that a software to help to find the best configuration of seats and tables in a room, with the goal of maximizing the number of people in that space, was a good software product.

2 Software

2.1 Functioning

The software takes in input a .json file containing a few software parameters and the restaurant map. The software after a while, depending on the size of the restaurant's map, returns possible seats configurations as restaurant's maps in .png format in output folder.

2.2 Technological specifications

The software is written in Python 3 and it is built up on CP-SAT[1], Google's constraint programming solver, part of the Google OR-Tools[2]. Other libraries used are: json, math, random, PIL and timeit.

2.3 Input file

As stated before the input file is in json format. It is composed of two parts:

[1]https://developers.google.com/optimization/reference/python/sat/python/cp_model

[2]https://developers.google.com/optimization

2.3.1 Software parameters

These parameters are values to control the execution of the program. They should be all present in the input file. They are:

- *seat_dim*: dimension of seats expressed in the same unit of other dimensions used in the software. It should be considered the edge of a square. If the seat is a round, consider this as the diameter.
- *time_limit*: if the map is very big, the solution can take a while, so you can set a time limit (in minutes) after which the software stops searching for other solutions and the best solution found up to that point is returned.
- *security_dis*: distance to keep between each seat. Keep in mind that distance between seats of the same table is not considered, as table-mates are seen as living together or close friends and for this reason social distancing is not a concern. Bear in mind that higher is the *security_dis* and less are the seats enabled in solutions.
- *save*: tells the program whether save or not the images to the output folder. The value can be True or False.
- *show*: tells the program whether show or not the images. The value can be True or False. Keep in mind that, if solutions are a lot, many images will be opened in different computer windows.

2.3.2 Map

To measure distances to create the restaurant's map you have to decide a fixed point in the room, our suggestion is to use the corner of a wall as a reference point but could also be a door or other stationary objects. From that point seats and tables should be measured in a cartesian plane way. To precisely measure distances, you have to consider the top-left corner of each object or better the closest corner to the reference point.

As we can see in Figure 1, table 1 position (in figure called x_1 and y_1) is calculated from the reference point to the top left, as well as the seat 4 position (in figure called x_2 and y_2).

To see a real example of the .json file containing parameters and the map see Section 3.

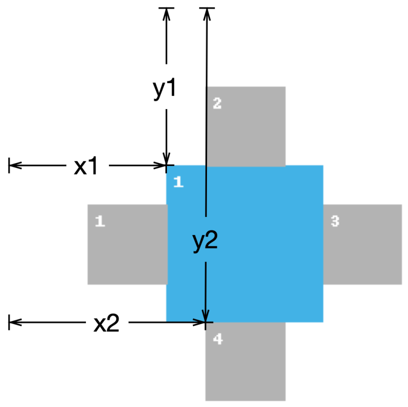

Figure 1: How to measure distances.

2.4 Software structure

The program is composed by three files. We will analyze them separately:

- *main.py*: This is only a start up file. There is a time calculation function to count the seconds spent by the solver to find the solution. Inside the "Editable software part" you can modify the code. By default a new instance of *Restaurant* is created and the *solve* method on that one is called. You can create as menu instances as you want, at each initialization the program asks for the .json input file. You can specify different one at each *Restaurant* initialization.

- *restaurant.py*: Contains the declaration of the *Restaurant* Class. The main core of the solver is contained in this file. See code documentation to understand better the functioning.

- *rmap.py*: Contains the declaration of the *RMap* Class. This is the way the software is able to draw the output map image. Also the show and save features are handled by this piece of software.

3 Sample case

To better understand the functioning of the software we will make a demonstrative usage of the program. First we have to create the .json file (you can find an example in the software source folder and you can use it as a template, called *input.json*). The file is shown in Figure 6. Seats dimension is considered 50cm, there is not time constraint so the program will output the optimal solution, security distance is here set to be 1.5 meters and the software will only save solution maps to the output folder and won't show them at the end of the execution. The program after a few moments will save to the output folder four maps, I report them in Figure 2, 3, 4 and 5.

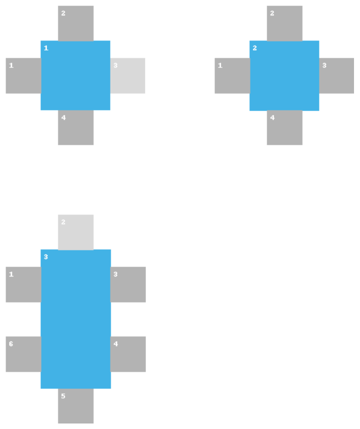
Figure 2: Solution 0

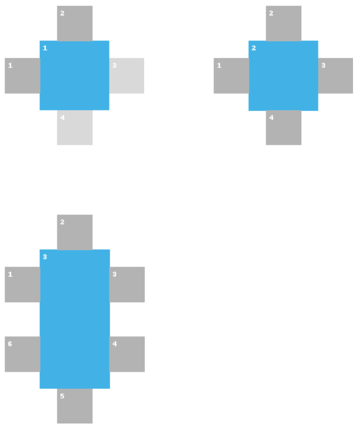
Figure 3: Solution 1

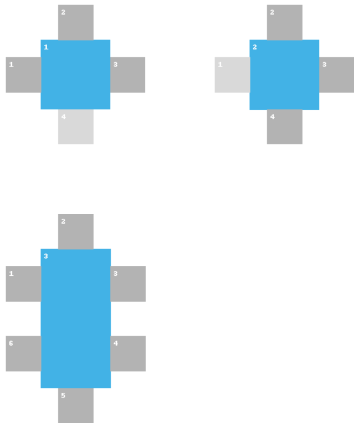
Figure 4: Solution 2

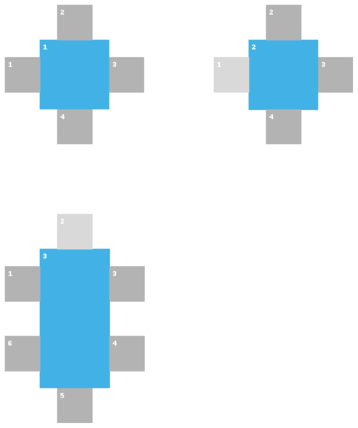
Figure 5: Solution 3

As you can see the program enable (dark gray) and disable (light gray) seats to maintain social distancing consistent with the *security_dis* parameter.

4 Details

4.1 Measures

I have decided not to use a fixed unit of measurement, the program is metric agnostic. In this way it can be used both with meters and feet. The only detail to keep in mind is that you should use numbers with a maximum of two decimals. Also consider that the default scaling factor in the *RMap __init__* method is set to 100. So, for example, 1.25 as a measure in the program will translate in 125px in the image and 0.01 will be only 1px.

4.2 Execution time

As stated before, if we have a complex restaurant map (e.g. many seats in different tables), the software could be very slow to find the optimal solution. To obtain a solution in a reasonable time you can set the *time_limit* parameter, but this way you will get a not optimal solution. The best way to solve the problem and obtain optimal solutions in a short time is to split your room in different maps and run the program for each of them. So you should consider different clusters of tables when it is clear that one cluster is far from the others. Let's think of two separate rooms, or two groups of tables separated by a large door.

4.3 Output files

If you set the parameter *save* to True, images will be saved in the output folder. Files will be named with the following scheme:

$$software\_execution\_id - solution\_id.png$$

Where *software_execution_id* is a random number between 0 and 1 million and *solution_id* is an incremental number from 0 to the number of solutions minus one.

5 Conclusion

I think that the program at this point can be considered usable to facilitate the organization of secure environment during this particular period. To further improve this software it could be added some form of advance constraints features to let the user filter the output solutions to have, for example, only a minimum number of seats for each table or other similar limitations.

A Appendix

```
{
  "seat_dim": "0.5",
  "time_limit": "0",
  "security_dis": "1.5",
  "save": "True",
  "show": "False",
  "tables": [
    {
      "x_pos": "1",
      "y_pos": "1",
      "x_dim": "1",
      "y_dim": "1",
      "seats": [
        {
          "x_pos": "0.50",
          "y_pos": "1.25"
        },
        {
          "x_pos": "1.25",
          "y_pos": "0.50"
        },
        {
          "x_pos": "2",
          "y_pos": "1.25"
        },
        {
          "x_pos": "1.25",
          "y_pos": "2"
        }
      ]
    },
    ...
    {
      "x_pos": "1",
      "y_pos": "4",
      "x_dim": "1",
      "y_dim": "2",
      "seats": [
        {
          "x_pos": "0.50",
          "y_pos": "4.25"
        },
        ...
        {
          "x_pos": "0.50",
          "y_pos": "5.25"
        }
      ]
    }
  ]
}
```

Figure 6: input.json example file