

אינטליגנציה חישובית – תש"פ סמסטר ב'

**פרויקט קורס**

Locating coronavirus testing centers as the capacitated

facility location problem with hard capacities

ארסני פרצובסקי - 317377372

רתם גליקי - 305328296

**Abstract**

Dealing with the coronavirus outbreak is still a main challenge for many countries, affecting almost every aspect of our lives. In order to cope with the pandemic, testing facilities that help detect and isolate infected individuals were needed to be opened. The decision of which testing facilities to open, where to open them and to which cities they will be assigned is a hard problem that can be modeled as the capacitated facility location problem with hard capacities. Five heuristics were implemented to solve this problem: Random Greedy, Constructional Greedy, Local Search, Simulated annealing, and Genetic algorithm. Results on different problem settings show a clear advantage for Local search and some potential for Simulated annealing for large number of iterations. Finally, we propose improvements based on assessing the model’s limitations and the performance of the implemented algorithms.

**Introduction**

Most of us will probably remember 2020 as the year of the coronavirus. From an innocent looking disease that seemed at the start like a distant cousin of the flu, it changed our everyday life and affected almost all aspects of it - employment, social life, vacations and more. Combination of high infection rate and no existing vaccine turned detecting and isolating infected individuals to one of the main strategies for coping with the virus. To make this possible, countries had to set up testing facilities, spread around the country, providing coronavirus tests for the local population. The decision of which testing facilities to open, where to open them and to which cities they will be assigned is a hard problem.

The problem presented in this project is the capacitated facility location problem with hard capacities. The settings include a set of facilities (F) and a set of cities (D) in a common metric space, built of N cells with location parameters (x,y). Each facility can be opened in one of the empty cells, making the problem discrete. Each facility i has a facility opening cost for opening the facility in cell m fim, and capacity ui that specifies the maximum number of city residents that may be assigned to this facility. Each city j has a total request uj, representing the daily number of city residents that need a coronavirus test. The total requests from all cities is higher than the total capacities of all facilities. We want to open all the facilities from the set F and assign residents from the different cities to an open facility so that at most ui residents are assigned to any open facility i. Hard capacities mean that facility i can be opened at most once to serve at most ui demand. We do not require that all demand from a city be served by a single facility, and each facility can serve multiple cities. The cost of assigning residents from city j to facility i is given by their distance cij, and the profit gained from each assigned resident is P. Because distance is not in measurement units as profit, we assume that each distance unit has a cost of one. Our goal is to maximize the total utility, given by the following equation:

The presented problem of capacitated facility location problem with hard capacities is an NP-hard problem. Hence, trying to find an optimal solution to the problem might require going over all possible solutions. Instead, using suitable heuristics can offer us solutions with good quality in polynomial time. The main challenge of generating dissent heuristics for solving the presented problem is the complexity of the required solution, reflected in the solution creating process and the numerous decisions that need to be made in order to create a valid solution. A valid solution requires first placing each facility in an empty cell, and then assigning them requirements from neighboring cities. The fact that the solution consists of these two different layers, makes it difficult to consider the outcome of every small change and how far or close it will take us from the optimal solution.

Our solution ideas spread from quick and simple algorithms to more complex ones, each with its benefits and drawbacks. The main flow of the different solution ideas is the same and is based on two phases - first place facilities, and then assign them with demands from the cities. The method of assigning cities to facilities is the same in all implemented algorithms. They differ from one another in the way the facilities are placed on the grid, the criteria for changing the current solution and more.

**Main part**

In total, five different heuristics were implemented to solve the capacitated facility location problem. The following section describes in detail the solution by each algorithm, from the simplest to the complex. The concept behind the following heuristics was to start simple, fast, and easy, and slowly add levels of complexity, in order to examine the balance between solution quality and running time over a wide range of optional algorithms. In all the following algorithms, a solution includes a location for each facility (given in a cell grid), and how much demand from each city is handled by each facility (given by an allocation matrix). When opening a facility or changing its location, the allocation matrix will be accordingly changed (by removing or adding demands).

**Random Greedy**

The first and most basic algorithm implemented was a random greedy algorithm. In each iteration the algorithm starts from scratch and creates a new solution - first, the facilities are randomly placed in empty cells, then demands from the cities are assigned to the placed facilities based on the distance between them (from the closest to the farthest). If the utility of the created solution is better than the utility of the previous solution, the current solution is saved. Basically, this algorithm is equivalent to randomly generating solutions (as the number of executed iterations) and choosing the best one.

**Constructional Greedy Heuristic**

Randomly generating solutions is simple, easy, and fast. The next algorithm does it a little bit differently. The facilities are placed one at a time. The algorithm randomly selects an empty cell for the current facility, and places it in the chosen cell only if it increases the total utility (after assigning cities to the added facility). In order to compare this heuristic to the others (that generate a valid solution every iteration), the described process occurs in each iteration, so that a new solution is created in each iteration. Same as before, If the utility of the created solution is better than the utility of the previous solution, the solution is updated to the current one.

**Local Search**

Local Search takes us out of the mostly random methods, and with some pre-planning can potentially help us reach improved results. This is a significantly more complex and structured algorithm, compared with the previous two. The neighborhood was defined as the solutions that can be reached by changing the location of only one facility. Because there are many possible locations to move each facility, only ten optional locations are tested in every iteration. That means, in each iteration ten randomly neighbors are selected, their utilities are calculated, and the one with the highest utility is selected to move to. If the best neighbor has a utility lower than the utility of the current solution, the solution remains the same.

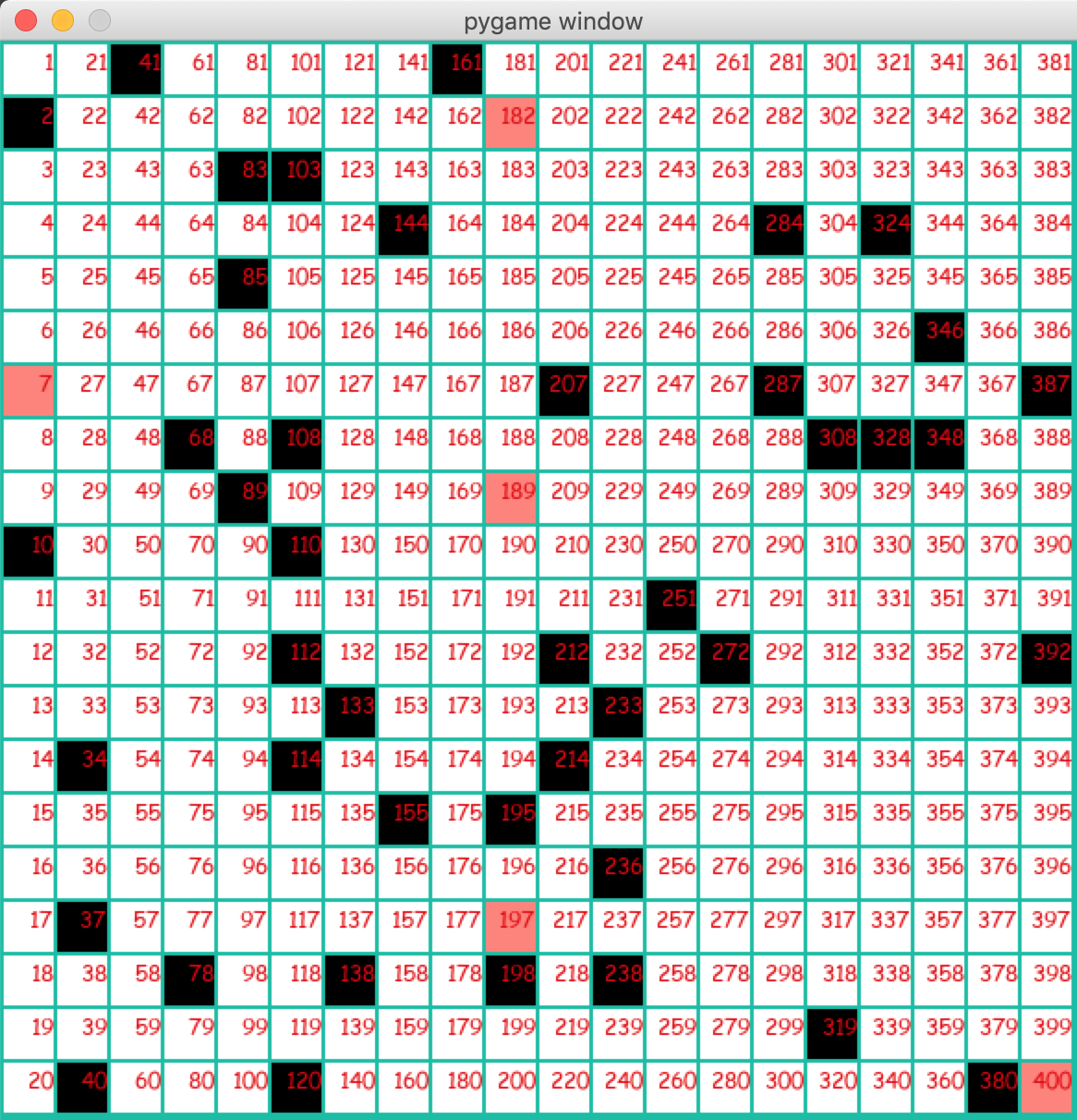
**Simulated Annealing**

Simulated Annealing was a natural continuation for us after local search, in which an exploration element is added to the algorithm in order to avoid local optimum. On the one hand, it may be time consuming (more iterations to get to the same result quality, due to downhill moves), but on the other hand it may bring better results, by escaping local optimum. The changes in regard to the implementation of local search are that in each iteration only one neighbor is randomly selected, and a move is made with a changing probability (depended on the delta from the current utility and the temperature function) The temperature function is a decreasing function, based on an opening temperature T0 = 100, and the formula .

**Genetic Algorithm**

The last implemented algorithm is the Genetic algorithm, that is inspired by evolution and the way it improves wanted characteristics over generations. In order to adjust the algorithm to our specific problem, we define the following: the population consists of ten items, each representing a solution for the problem. In each iteration a new population is created from the previous one using the following steps: calculate the utility for each item in the current population, according to the utility calculate the items reproduction probability, choose ten times a couple of items with the calculated probability, for each couple create a child (the item in the new population). A child is created by copying the location of half of the facilities from one parent, and half from the other. After the facilities are copied from the parents to the child, the cities are allocated to the facilities based on the distance between them (from the closest to the farthest).

In order to test the different algorithms on the chosen model a simulator was created using the PyGame package. PyGame imitates the sites and facilities in a 2D grid. In addition to running the desired tests, the simulator creates a for each iteration a graphic representation of the current selected solution, as shown in Figure [1]:



**Figure [1]: red squares - facilities, black squares - cites**

After defining the values for the test parameters, the simulator creates a random problem with the desired settings, and then runs all the algorithms on the same problem (to decrease the variance). Afterwards a new problem is generated, each problem with a different set of city-locations demands and costs. During the run of the simulator, the utility value per algorithm, per problem and per iteration is stored in order to be able to later calculate the needed statistics on the data and for plotting the results. Each point on the subsequent graphs is an average of N problems given an iteration and an algorithm.  All the code is represented in the following repository:

<https://github.com/Arseni1919/computational_intelligence_course_task>

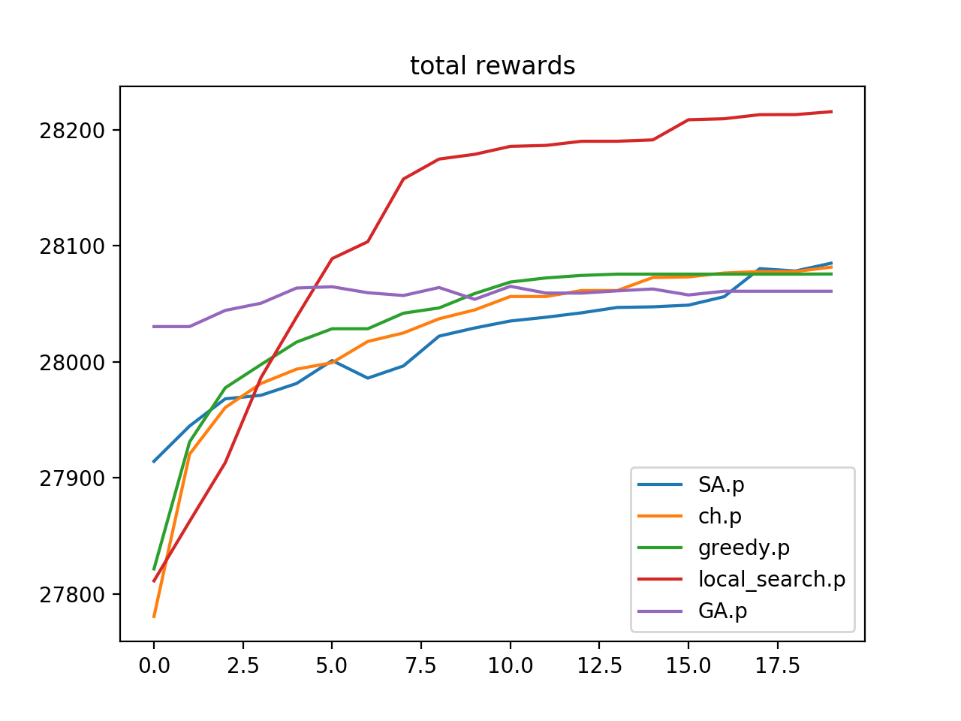
The chosen heuristics were tested and compared by value (utility) per iteration over two sets of experiments with the following settings: 20X20 grid cell, 5 facilities and 10% or 30 % cities (from the total number of cells). The demand for each city and the capacity of each facility was sampled uniformly between 10 and 100. In each setting 15 different random problems were generated, and on each problem, we ran all five algorithms.

**Results**

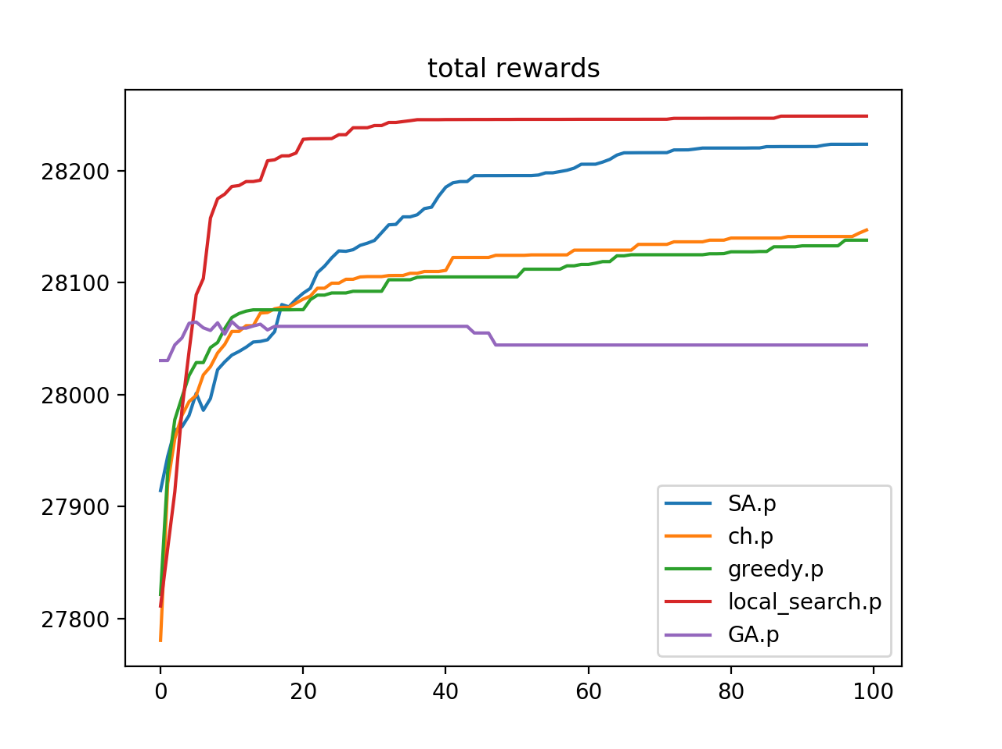
Figures 1 to 4 show the results of our experiments, with the settings as specified in the previous chapter. The results for city ratio of 10% and 30% are similar, with more diversity in the solution quality per iteration in the experiments with 30%. At first view, we can see a distinct advantage for Local Search (LS) over the rest of the implemented heuristics. As observed in all the graphs, LS shows rapid improvement rate and reaches quite fast high-quality solutions, then slowly keeps improving with a much smaller improvement rate, to the point of stabilization. At the other end, Genetic Algorithm (GA) struggles to improve and even deteriorates over time. It seems that from different reasons GA is less suitable for our problem (a complex algorithm with rather poor results). One possible problem could be that the method of creating a new child from two chosen parents while copying their facility locations, does not necessarily generate an improvement from generation to generation.

When looking at the performance of Random Greedy (RG) and Constructional Greedy Heuristics (CG) over the different experiments, the results are quite the same and with medium success. After a short period of fast improvement, the algorithms stable and continue very slowly to improve. In the end, both heuristics are based mainly on random elements for creating a new solution (random locations to one or all the facilities), which limits the algorithm's ability to improve. Overall, with fast and simple solution ideas we were able to achieve fair results. Last but not least, Simulated Annealing (SA) demonstrates steady improvement over time, although several drops in the solution quality due to exploration steps. In graphs 1 and 3 (total 20 iterations) it is clear that SA hasn’t reached stability yet and still has a significant improving potential. This is validated in graphs 2 and 4 (total 1000 iterations), where while LS reaches good results very fast and then stables, SA improves more slowly over time and reaches at the end almost the same result.

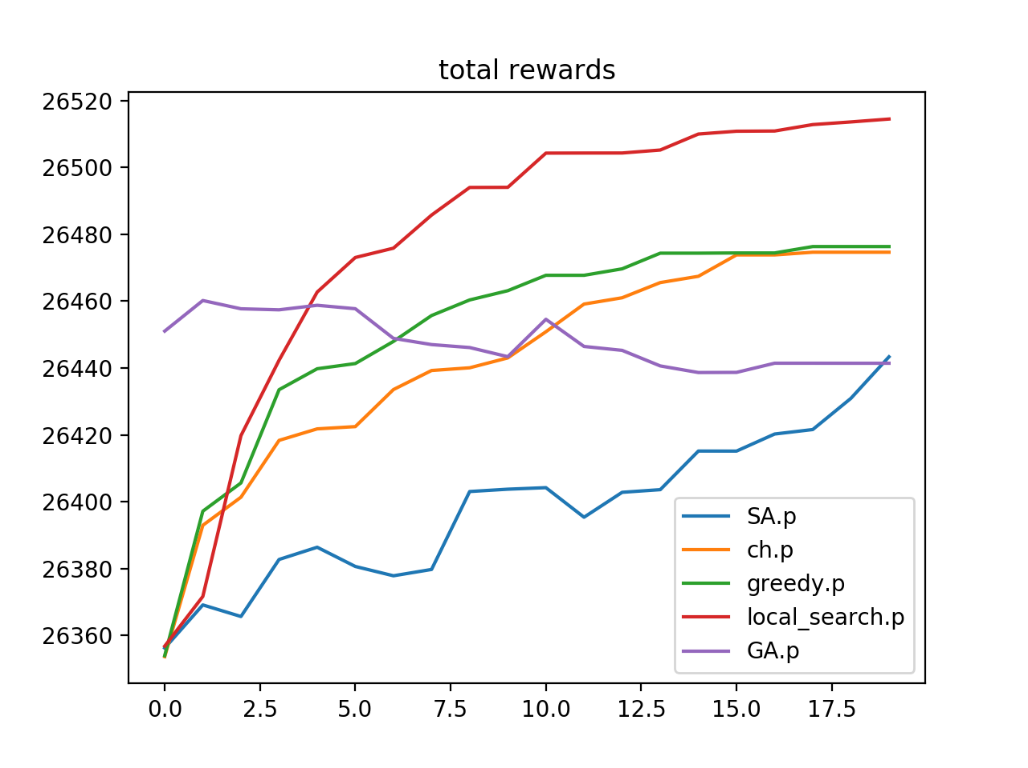
In an overall look, LS provides the best results in the tested settings and reaches great solutions very fast. However, it is possible that one of the other heuristics (probably SA) may reach even better solutions if given enough time (iterations). This is an important issue to pay attention to - there can be many criteria for choosing a heuristic to use when solving a hard problem, solution quality is just one of them. Running time, sensitivity, and the number of iterations until achieving a satisfying result are examples for other criteria. Accordingly, there cannot be only one answer to the question which heuristic is preferred, and it directly relates to the criteria we choose and their importance. That being said, for settings such as the experiments shown in this section, we recommend using LS for the capacitated facility location problem. If possible, in terms of running time, SA should also be considered (may reach better results for a large number of iterations). Moreover, it is interesting to see that more complex algorithms do not necessarily lead to better results.



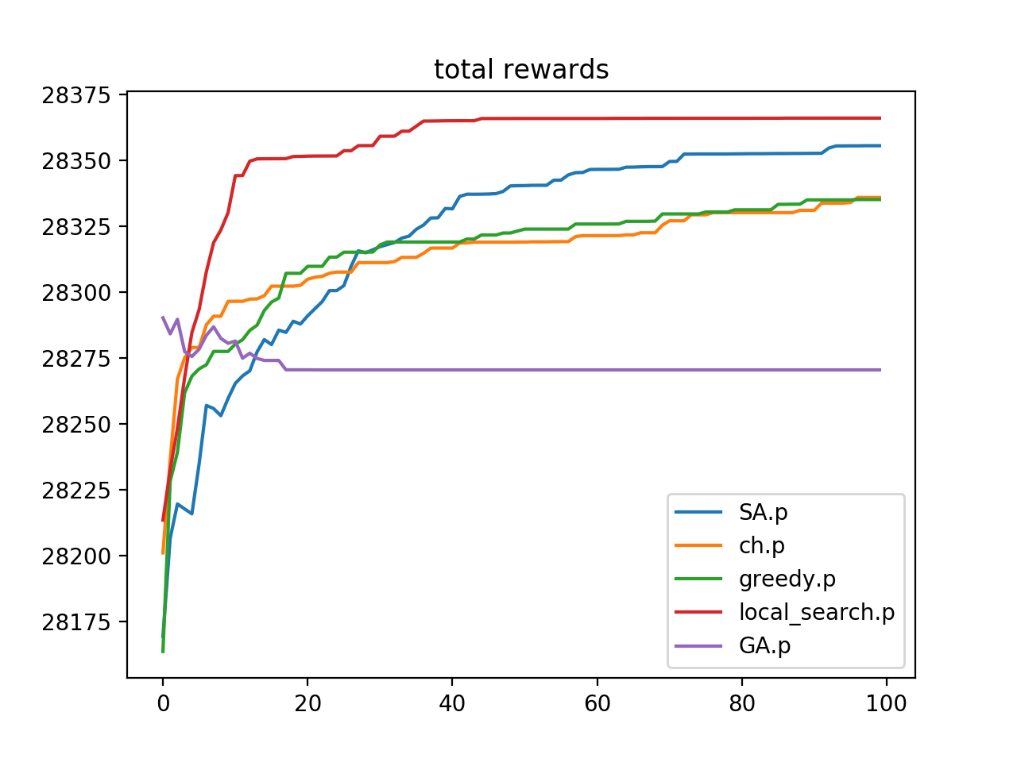
GRAPH [1]: **20** iterations, 5 facilities, ratio of **0.1**, each point of each graph is an average on 15 problems



GRAPH [2]: **100** iterations, 5 facilities, ratio of **0.1**, each point of each graph is an average on 15 problems



GRAPH [3]: **20** iterations, 5 facilities, ratio of **0.3**, each point of each graph is an average on 15 problems



GRAPH [4]: **100** iterations, 5 facilities, ratio of **0.3**, each point of each graph is an average on 15 problems

**Discussion**

The described problem of locating coronavirus testing facilities and assigning them with demands from the surrounding cities, is a real and relevant problem. It is a main part of the government's mechanism to manage an unfamiliar disease that knocked on our door one day in early March and does not show any signs of leaving in the near future. The chosen way of solution has great financial and health repercussions, so finding the best solution we can (in the time, computational and other limits, of course) is of high importance.

Obviously, no model can fully represent or match reality, and thus has its limitations. One of them refers to the uncertainty in real life - although the number of requests for coronavirus tests in each city changes daily, the model assumes this number is fixed. When using the model this number could be for example the mean of requests over the last month, but when applying the solution given by one of the algorithms one should remember that due to these changes in demands the solution may be good some days and less in others. Practicality is another limitation of the model that meets us when trying to execute it on a real problem. In the described model, the main limitation of this sort is that the world is represented by a grid cell. Since Israel is unfortunately not a grid, some conversion must be made (using some sort of map of actual city locations with several possible locations for facilities).

Other limitations of the model refer to how close are the problem settings to reality. One way to make the model closer to reality runs through the calculated elements of the model - we could use a more realistic distance function that is based on an actual road map instead of using air distance. Another way is to improve the accuracy of the already used parameters in the model, like calculating a more realistic travel cost per distance (when calculating the cost of assigning a request from a city to a facility) instead of assuming it is equal to one. In addition, more real-life elements could be added to the problem description. For example, we could add to the grid locations to stay away from (such as nursing homes, to avoid infecting the elderly) or add locations to be closer to (such as the testing laboratories for coronavirus).

Besides improving the model, improvements could be made also in the proposed heuristics. After reviewing the performance of the implemented heuristics over different settings, we believe that several improvements can be made in order to achieve higher rewards. First, we propose an adjustment relevant to all the algorithms - changing the inner method of assigning cities to facilities (after locating them) from greedy based to a more sophisticated one. This will allow us to reach higher rewards within the same facility locations. Another improvement to be considered is changing the way information is propagated from parents to their child when creating a new population in GA. Instead of copying just the locations of the facilities from each parent, we could also retain some of the information about the cities that were allocated to it (and not allocating the cities from scratch).

The randomly greedy heuristic could be improved by taking into consideration the spread of the cities on the grid cell when selecting the cells in which to open facilities. For example, we could divide the grid to sections, and tie between the number of cities in a section to the probability a cell from that section will be randomly selected for opening a facility. That means, the more cities a section has, the more likely facilities will be opened in it. Hopefully, this will improve the solutions by avoiding opening facilities far from cities. Constructional greedy heuristic could also be improved by adding an exploration element to it. This could be done by making it epsilon greedy and not entirely greedy - add facilities even if they do not increase the utility with some probability.

Another way to improve the algorithms is simply combining them. For example, running RG for 20 iterations, and use the best solution found as the starting point of LS or SA. We could also run CG for 10 iterations and use the generated solutions as the starting population in GA. Other improvements could be achieved by tuning the parameters used in the algorithms. This includes for example: population size and mutation probability in GA, temperature function in SA, the number of neighbors examined in LS etc. The process of improving the parameter values could be done using trial and error, based on prior theoretical assumptions or machine learning.