Using an SMT Optimizer to Solve Spatial Planning Problems: A Case Study on Installing COVID-19 Test Centers

Yichen Aaron Huang

September 3rd, 2021

1. Introduction

1.1 Background and Rationale

Spatial planning problems in urban development almost always entail large quantities of data and heavy computational work due to the need to search for the best solution out of an array of different possibilities (Mitchell; Chakrabarty). This makes computer assistance almost inevitable when looking for optimal solutions in spatial planning. While spatial planning in urban development may cover a wide range of topics, this paper will specifically focus on the issue of installing new infrastructure/facilities as amenities. The goal of this project is to utilize satisfiability modulo theories (SMT) to maximize resource-efficiency in planning problems. In this paper, resource-efficiency is defined as maximizing the accessibility of the newly installed public facility while spending a set amount of resources. In other words, the question this paper is asking is how can a planner satisfy the greatest number of people by decreasing their travel time while building a limited number of facilities/infrastructures?

This paper uses the case study of installing Coronavirus disease 2019 (COVID-19) test centers to test the algorithms designed. COVID-19 is a highly contagious respiratory disease caused by the SARS-CoV-2 virus. As of July 30th, 2021, more than 200 million cases have been confirmed worldwide according to the World Health Organization (WHO). With testing being highlighted as an essential way to tackle the virus by the WHO, the accessibility to COVID-19 test centers is essential. Not only does the ease of access to test centers increase the ease to gain a travel permit for residents of certain nations, it is also an effective means to tackle the virus as a whole by identifying more patients. Since resources for opening test centers are always finite, it is essential to maximize benefits while being intentional about spending resources. Thus, this research paper attempts to use SMT solvers to maximize the accessibility of COVID-19 test centers while building a limited number of these facilities.

1.2 Tools & Materials

First, this project uses the Z3 Theorem Prover created by Microsoft to solve the problem. While Z3 is an SMT solver, which generalizes boolean satisfiability (SAT), it can also be used as an optimizer with the extension *vZ* (Bjørner et al.; de Moura, Leonardo and Bjørner). *vZ* is an optimizing SMT solver that allows users to solve SMT constraints while computing for optimal variable values. The *vZ* solver is used to optimize the minimizing/maximizing problem of resource-efficiency in urban development. The coding for the *vZ* solver is done exclusively through Python, a high-level general-purpose programming language, in conjunction with the Python library *Z3*-

Solver (Kuhlman). The Python library *Z3-Solver* allows a more convenient way to create SMT constraints and variables in the Z3 solver by specifying a custom amount.

Second, this project uses OpenStreetMap (OSM), an online data collection of the world map detailing buildings, public transit, roads, and many more features. According to its official wiki, OSM uses crowdsourced information as its data, including surveys, aerial photographs, and users' own local knowledge ("About OpenStreetMap - OpenStreetMap Wiki"). While not every infrastructure is registered or registered correctly on OSM, the data can still serve as a fairly reliable source to model a city with. To read the data from OSM, this project uses Python and its library *PyOsmium*.

Lastly, this project uses two additional Python libraries to organize the data. The Python library *Pandas* was used when reading data from OSM for better storage and manipulation. More specifically, after *PyOsmium* unpacks the data into a readable format, *Pandas* is used to store the data into an easier structure to manipulate with. Additionally, the Python library *Matplotlib* was used to generate the graphs based on computed results.

1.3 Related Works

Although no existing literature has created the same solution this paper offers, similar problems have been tackled before. Joao F. M. Sarubbi et al. from the Federal Center for Technological Education of Minas Gerais in Brazil have tackled a similar problem, the School Bus Routing Problem, or SBRP (Sarubbi, Joao F. M et al.). In their paper, they attempt to minimize school bus travel time while assuring that every student will have a bus stop at a reasonable distance (a configurable variable) from their house. The main difference between the research on SBRP and this paper is the goal of the solver. While SBRP seeks to minimize the bus travel time, this paper attempts to maximize the accessibility for the general population. In an SBRP, bus stops are opened based on their effect on the total travel time of the bus. In the problem illustrated in this paper, COVID-19 test centers opened based on how it will impact the population's travel time/how many people can reach it in a reasonable distance (see 2. Methodology).

Other works using computer optimizers or solvers as a means to combat COVID-19 also exist. The most notable example comes from the research published by Christian Alvin H. Buhat et al. in the *Journal of Healthcare Informatics Research*, or JHIR (Buhat et al.). These researchers attempted to create an algorithm to determine which hospitals should be allocated the most COVID-19 test kits based on the distribution of the infected population. While this paper does not take into account different levels of COVID-19 severity in different regions, the results can be combined with the paper from JHIR to produce a more realistic solution to installing COVID-19 test centers. As the solvers built in this paper aim to determine which COVID-19 test centers should be opened, the solvers designed by the paper from JHIR may be used to amend the product by determining how many testing-kits each center should receive.

2. Methodology

2.1 Constructing the Problem

The accessibility to test centers depends on a variety of variables; however, when it comes to choosing where to open them, the distance of travel is often the main deciding

factor. By minimizing the travel distance required for a person to get tested, the accessibility will be maximized.

To simulate the problem, a model of a city is constructed. First, this project records all the residential buildings and counts them as one unit, meaning that every household is treated equally. The rationale for this approach largely comes from the fact that not enough data is present to detail every single individual's location of occupation; therefore, this project chooses to use a household in place of a person. Although households may differ in size, when the scale grows large enough, for example the scale of a city, these minute differences may be ignored. Then, this project gathers all the possible locations for opening a test center to determine the optimal way of opening them. Since testing may happen in all sorts of facilities ranging from hospitals to parks, it is hard to determine which location is a feasible spot. Therefore, this project simply uses all supermarkets to represent potential test center locations. The benefit of using supermarkets is that the data for supermarkets is relatively well registered on OSM. Also, supermarkets are usually quite spread around the city; thus, they serve as an effective way to mimic real-life potential locations for opening a test center. Last, the distance between households and test centers is calculated by measuring the straight-line distance. Although driving distance or traveling distance may not equate to straight-line distance, in a city with a developed road system, the two metrics are similar enough to not affect the result. In fact, research by Francis P. Boscoe et al. published by the New York State Cancer Registry concluded that when calculating driving time to a hospital in a non-emergency situation, using travel distance and straight-line distance hardly makes a difference (Boscoe et al.).

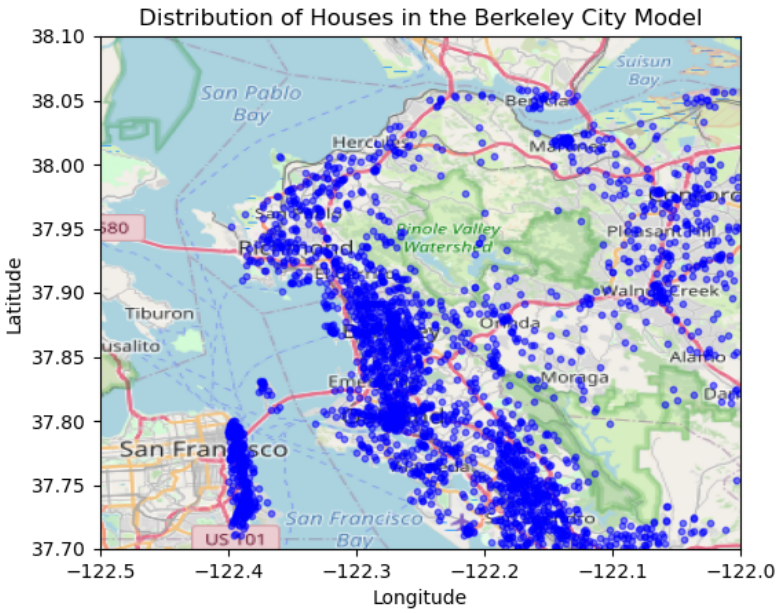
To solve the problem, two methods are attempted in this project. The first method was designed to minimize the total travel distance by all the households assuming only x test centers were opened. However, using such methods may favor households in areas where the population is dense while neglecting households situated in the outskirts of the city. The second method was designed to maximize the percentage of households able to travel to a test center within d distance assuming only x test centers were opened. This method allows the creation of a way to ensure that 100% of the population can reach a test center within d distance by manipulating with d and x.

2.2 Reading the Data

This project uses georeferenced data of real cities gathered by OSM: in particular, the data of Berkeley City and Ghent. Although the data is titled Berkeley City and Ghent, the actual area is much larger than the respective cities themselves. The Berkeley dataset for example, covers a significant portion of the San Francisco Bay area rather than only the city itself. In fact, the area covered is estimated to be about 1,936 square kilometers. The main reason why two locations were chosen is because it allows testing the algorithm with two different datasets, which would make sure that certain problems are not caused by the nature of one dataset.

The location of the buildings read are stored as latitudes and longitudes. However, due to many residential buildings not being registered in the OSM database, this project counts all buildings as a unit of household whether or not they are residential. This approach allows the creation of 82,493 households instead of 1,725 (the number of

registered residential buildings) when using the data of Berkeley City. The number 82,493 is a much more reasonable number for an area of 1,936 km^2. Moreover, the OSM database does not include the number of stories for buildings; therefore, large residential buildings such as apartments may be undervalued in this project. Yet the data gathered can still be used as an effective tool to experiment with the algorithms created in this file, even though the modeled city might not be an accurate representation of a real city.



Each dot represents a building, only one out of thirty buildings were plotted

With the use of *Pandas* and *PyOsmium*, the data from OSM was stored in two text files: the first for all the buildings and the second for all the potential locations for test centers. The main reason why the data is first stored in a text file is that the reading of OSM data is a slow process, suggesting that if the reading, preprocessing, and solving were all to be done in one file, it would be too slow for the experimental process.

Based on the OSM database, the model for Berkeley City contains 82,493 buildings and 79 potential locations for test centers, while the model for Ghent contains 43,741 buildings and 106 potential locations for test centers.

2.3 Preprocessing

The initial number of buildings and test centers is too large for an algorithm to run efficiently; therefore, this project preprocessed the buildings into clusters in two ways to suit the two methods used.

The entire process of preprocessing was done in Python with Python variables. The reason is that the *Z3-Solver* is particularly slow and difficult to work with when manipulating variables. As an SMT solver and optimizer, Z3 is efficient at computing an SAT result but inefficient at regrouping variables or assigning variables with certain values.

2.3.1 Minimize Total Distance

To group buildings into clusters, a commonality must be found first. Therefore, all the buildings were assigned a ranked list of the closest test centers to them. All buildings with the same list of ranking were grouped into clusters, and that cluster will now hold the list of ranking along with the total distance of travel from all the households. Consider the example below.

```
Test_Center = [A, B, C]
//there are three test centers: A, B, and C
Rank_Of_House1 = [A(5), B(6), C(8)]
//A is closest to house1 at 5km
//B is the second closest, at 6km
Rank_Of_House2 = [A(8), B(10), C(11)]
Rank_Of_House3 = [C(3), A(7), B(9)]
```

These three houses will then be put into two clusters. $Cluster_{ABC}$ would be the combination of $house_1$ and $house_2$, meaning that these two houses would be treated as the same house when implementing the solution.

```
Cluster_ABC = [13, 16, 19]
//this cluster is the combination of house 1
and 2
//13 is in the 0th index, since A (the closest
test center) would require 5km for house1 to
reach and 8km for house2 to reach
Cluster_CAB = [3, 7, 9]
```

After using this method of preprocessing, the number of buildings in the Berkeley City model went from 82,493 to 41,135, essentially halving the number. The number of buildings in the Ghent model went from 43,741 to 35,889. The main reason why Ghent did not experience much of a drop is that the number of test centers in Ghent is slightly higher than Berkeley, while the number of houses is significantly lower. This means that more possibilities of ranking exist in Ghent while fewer houses would occupy each kind of ranking, making this preprocessing not as effective when compared to Berkeley.

2.3.2 Maximize Reachable Population

The second way of grouping buildings into clusters is by assigning each building an array of the test centers within *d* distance. All buildings with the same array of test centers were grouped into one cluster, making them essentially one building. Consider the following example;

```
Test_Center = [A, B, C]
//there are three test centers: A, B, and C
Array_Of_House1 = [A, B]
//both A and B are within d distance of house1
Array_Of_House2 = [A, B]
Array_Of_House3 = [C, A]
```

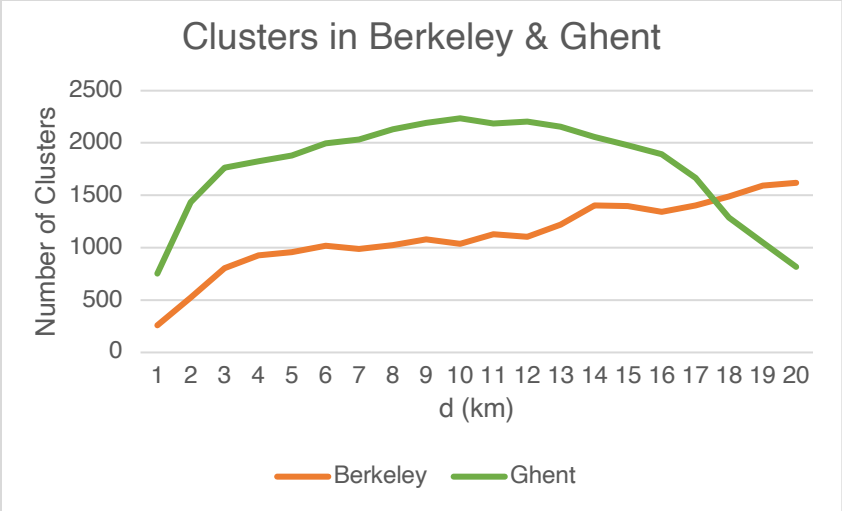
These three houses will again be put into two clusters. $Cluster_{AB}$ is the combination of $house_1$ and $house_2$ since both of them can only reach test centers A and B within *d* distance. $Cluster_{AB}$ would then be assigned the value of 2 since 2 houses satisfy its condition.

```
Cluster_AB = 2 //two houses: house_1 + house_2
Cluster_CA = 1
```

When *d* is set to 7 kilometers, this method reduced the number of buildings in the Berkeley City model from 82,493 to 986. The number of buildings in the Ghent model went from 43,741 to 2,033. This method of reducing the amount of data is much more effective when compared to the previous method, mainly because the number of possible clusters is significantly reduced. Not every test center is within *d* distance of all the houses; therefore, many clusters only consist of houses that can reach one or two test centers. This allows the second method to reduce the data much more effectively when compared to the first. However, the same problem with Ghent occurs in the second method as well. As more possibility of clusters exists with fewer houses to fill those clusters, Ghent experiences less of a drop when compared to Berkeley.

d (km)	Clusters in Berkeley	Percentage Drop in Berkeley	Clusters in Ghent	Percentage Drop in Ghent	Average Percentage Drop
3	808	98.16	1766	95.97	97.07
6	1021	97.67	1996	95.44	96.56
9	1081	97.53	2195	94.99	96.26
12	1105	97.48	2207	94.96	96.22
15	1395	96.82	1976	95.49	96.16
18	1488	96.6	1285	97.07	96.84

Based on the data collected, as *d* rises initially, both models experienced an increase in the number of clusters. However, Ghent saw a gradual decline after the number of clusters peaked at around 2,200 with *d* at around 12 kilometers. Berkeley, on the other hand, saw a continuous rise in the number of clusters during the incrementation of *d* all the way to 20. Although this finding does not have a direct impact on the result of this paper, it is still interesting to note how different geography and locations of potential test sites can impact the formation of clusters. More importantly, this finding shows a difference between Berkeley and Ghent, which validates the algorithms described later by showing their applicability to different environments.



2.4 The Solutions

All the optimizing (minimizing and maximizing) process is done with Z3's optimizer *vZ*. The reason is that *vZ* uses methods such as backtracking when optimizing, which is significantly faster than conventional methods such as greedy or brute force. Although other solvers may use heuristic methods that would greatly increase the speed further, heuristic approaches do not always guarantee the optimal solution.

2.4.1 Minimize Total Distance

To minimize the total travel distance, the distance traveled by each cluster must be calculated. First, a boolean variable was created for every test center to represent its state: either opened (True) or closed (False). Then, depending on which center is opened, each cluster will have a total distance traveled value selected. The one selected will always be the value between this cluster and the closest test center that is open (True). Additionally, a constraint of $\leq x$ is set on the total number of test centers that can be opened (True). This way, the program may take input data detailing the number of test centers that can be opened. Furthermore, more test centers usually lead to more people being able to access test centers; thus, the number of opened centers will most likely equate to x. Last, a variable *total_distance* is created to sum up the distance traveled by all the individual clusters. These variables and constraints are then fed to Z3 with the goal of minimizing *total_distance*. Consider the following example;

```
x = 1
//only one test center may be chosen
Test_Center = [A, B, C]
//there are three test centers: A, B, and C
Cluster_ABC = [13, 16, 19]
//this cluster contains house_1 and house_2
Cluster_CAB = [3, 7, 9]
//this cluster contains house_3
Total_distance = 0
If (A): Total_distance = Cluster_ABC[A] +
Cluster_CAB[A] //13 + 7 = 20
//if A is chosen, then Cluster_ABC will have a
travel distance of 13 and Cluster_CAB will have a
travel distance of 7
Else If (B): Total_distance = Cluster_ABC[B] +
Cluster_CAB[B]//16 + 9 = 25
Else If (C): Total_distance = Cluster_ABC[C] +
Cluster_CAB[C]//3 + 19 = 22
```

The end result Z3 would produce is that the optimal location to open is test center A and the total distance traveled would be 20, meaning that the average distance traveled per household would be 20 divided by 3 (the total number of households).

2.4.2 Maximize Reachable Population

Similar procedures happen in this method when compared to the previous. Every test center also receives a boolean variable that would represent its opened (True) and closed (False) state. A variable *number_of_people* is then created to tally how many people can reach a test center within *d* distance. If a cluster has one or more of its reachable test centers open (True), then the number of households this cluster represents will be added to *number_of_people*. As before, a constraint of $\leq x$ is set on the total number of test centers that can be opened (True). Since opening test centers will allow more people to reach test centers, the number of opened (True) test centers will almost always equate to x. These variables and constraints are then fed to Z3 with the goal of maximizing *number_of_people*. Consider the following example;

```
x = 1
//only one test center may be chosen
Test_Center = [A, B, C]
//there are three test centers: A, B, and C
Cluster_AB = 2 //two houses: house_1 + house_2
Cluster_CA = 1 //one houses: house_3
Number_of_people = 0
If (A or B): Number_of_people += Cluster_AB //2
//if either A or B is open, then the 2
households in Cluster_AB may reach a test
center. So the Number_of_people += 2
If (C or A): Number_of_people += Cluster_CA //1
```

Z3 would choose to open test center A since that would allow all three households to reach a test center within *d* distance, which would translate to a 100% satisfiable rate.

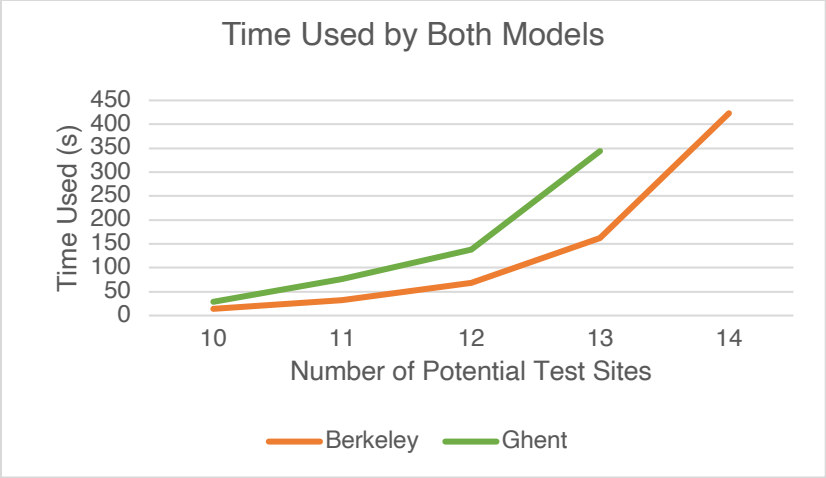
3. Experiment

When applying the data of both Ghent and Berkeley, the numbers were too large for both solutions to handle even after the preprocessing. Although both algorithms were unable to handle a large amount of data for different reasons, both of the issues stem from a large number of potential test sites. Take Berkeley as an example. Assuming x, or the number of test centers that can be opened is five, then the total number of possible solutions would be ${}_{79}C_5$, or $2.2537515 * 10^7$. Therefore, when testing with these algorithms, the number of test centers was lowered and the locations were randomly chosen.

3.1 Minimize Total Distance

With ten potential test sites, preprocessing was able to reduce 82,493 houses in Berkeley City to 303 clusters. Using those clusters and with x set to five, the minimum average distance a citizen in the Berkeley City model will have to travel is 7.096 kilometers. When the same experiment was conducted on Ghent, 559 clusters were formed assuming x was set to five, and the average travel distance in the Ghent model was 3.858 kilometers. It is quite interesting how the average travel distance in Ghent is significantly less than Berkeley. This is likely due to where the potential testing sites were placed.

When attempting to increase the number of potential test center locations, the algorithm grew drastically slower. When testing, we aborted every program that lasted over ten minutes. For the Berkeley City model, ten potential test center locations ran for 14.8 seconds, with 1.08 seconds in preprocessing and 13.72 seconds in solving. Yet the program failed to complete under the ten-minute mark when the potential test center locations were increased to fifteen. The previous attempt with fourteen potential test locations lasted for 423.16 seconds, which translates to about seven minutes. The preprocessing took 1.42 seconds while the solving took 421.74 seconds. On the other hand, the Ghent model used 28.82 seconds when computing for a solution with ten potential test centers, with 0.59 seconds in preprocessing and 28.23 seconds in solving. The maximum number of test centers the Ghent model was able to compute is thirteen. It took the model 344 seconds to finish computing. Despite the number of test centers only receiving a small amount of increase, the time the program takes increases drastically. This is likely due to the exponential scaling this method has with the number of clusters.



As the Berkeley model went from ten test centers to fourteen, the number of clusters went from 303 to 1,148. The increase in the number of clusters drastically outweighs the amount of increase in the potential test locations. Since every cluster would require the number of if statements equivalent to the number of test centers, increasing from ten to fourteen increased the number of constraints from 3,030 (303 multiplied by 10) to 16,072 (1,148 multiplied by 14). A forty percent increase in test center locations resulted in almost a 430 percent increase in the number of constraints. Moreover, as the number of test centers increases, potential solutions also increase from ${}_{10}C_5$ to ${}_{14}C_5$, or 252 to 2,002. Therefore, although the increase in data is linear, the increase in the computational work for the algorithm is not. Additionally, the running time is also effected by some of Z3's shortcomings when fed too many variables. Since the *vZ* optimizer forces itself to look for the optimal solution, a lot of time will be spent on validating whether or not the solution is the optimal.

Although the number of test centers will drastically affect the running time of this method, the number of houses affects it to a lesser degree. When the number of houses in Berkeley was reduced from 82,493 to 1,725, the method was still unable to finish computing in time with all 79 test centers in Berkeley. The reason is that the number of clusters formed mainly depends on the number of potential test centers, since more test centers means more ways of ranking them, and the clusters are formed based on the house's rankings on the test centers. This is demonstrated by how even though the number of initial houses was reduced to 1,725, there was still 1,088 clusters. Something worth noting is that the number of clusters with the smaller model (1,088) is less than the original model with 14 potential test locations (1,148); yet, the algorithm still failed to finish computing within time. This suggests that other variables also have an effect. The most likely variable to impact the running time is the number of test centers itself, as it is also a factor that impacts the number of constraints entered into Z3 and the number of potential solutions. Therefore, when applying this algorithm in real life, the number of buildings or the size of the city will not make much of an impact as long as there is a controlled number of potential test center locations.

3.2 Maximize Reachable Population

For the Berkeley model, 49 clusters were formed under the constraint that x is set to five and d is set to seven kilometers. The result shows 64.11% of the population able to reach a test center within seven kilometers. As for the Ghent model, this algorithm generated 52 clusters with x as five and d as seven kilometers. 89.06% of the population can reach a test center in Ghent, which is significantly higher than Berkeley.

This algorithm used ten potential test centers to serve as a comparison with the previous model. Yet this algorithm is capable of running test sizes far larger than ten potential test centers. The following chart details time of running with different test centers under the constraint that x is five and d is six kilometers.

Number of test centers	Preprocessing Time (s)	Solving Time (s)	Total Time (s)
10	0.86	0.02	0.88
20	1.69	0.11	1.8
30	2.53	0.28	2.8
40	3.33	0.59	3.92
50	4.18	42.06	46.24
60	5.17	182.4	187.57
70	5.84	TLE	TLE
79	6.61	TLE	TLE

**TLE = TIME LIMIT EXCEEDED*

Even though the algorithm exceeds the time limit at 70 test centers and onward, that is due to the special nature of having d set to six kilometers. As shown by the chart below, 70 test centers can be handled as d changes.

Number of test centers/*d*(km)	3	6	9	12	15	18
10	0.86	0.88	0.89	0.88	0.88	0.9
20	1.79	1.8	1.75	1.75	1.82	1.81
30	TLE	2.8	2.67	2.74	2.72	2.75
40	TLE	3.92	3.59	3.61	3.74	3.74
50	TLE	46.2	4.67	4.58	4.74	4.82
60	TLE	188	5.83	5.45	5.83	5.85
70	TLE	TLE	6.59	6.49	6.93	7.06
79	TLE	TLE	7.3	7.51	7.99	8.28

**TLE = TIME LIMIT EXCEEDED*

The data in this graph records the total amount of time used to run the algorithm as d and the number of test centers changes. When d is set to three and six, the algorithm performs extremely poorly due to an issue with *vZ*. Although the exact reason remains unclear, it is likely due to *vZ*'s nature that forces it to find the optimal solution. Our hypothesis is that certain numbers may produce unorthodox combinations of constraints that could drastically slow down the process by forcing *vZ* to look at every possible solution. This hypothesis is reinforced by how the algorithm spent 188 seconds when the

number of test centers is 70 and *d* is six while only spending 6.59 seconds when *d* is nine. The chart below shows the number of clusters as *d* and the number of test centers changes.

Number of test centers/*d*(km)	3	6	9	12	15	18
10	30	45	58	50	5848	50
20	86	121	164	156	168	181
30	155	227	292	288	317	328
40	297	394	445	449	503	502
50	441	596	611	621	743	716
60	540	694	724	730	918	901
70	690	868	890	905	1137	1175
79	808	1021	1081	1105	1395	1488

**Shading = TIME LIMIT EXCEEDED*

Even though the number of clusters when *d* is three is smaller than many other situations, the algorithm still fails to compute a result under ten minutes. This again shows affirmation for our hypothesis as even though the number of clusters is smaller in certain cases, it still fails to compute in time while others can compute much faster with more clusters. The same problem occurs in the Ghent Model as well. The chart below shows the time taken in seconds for the Ghent Model as *d* and the number of test centers changes.

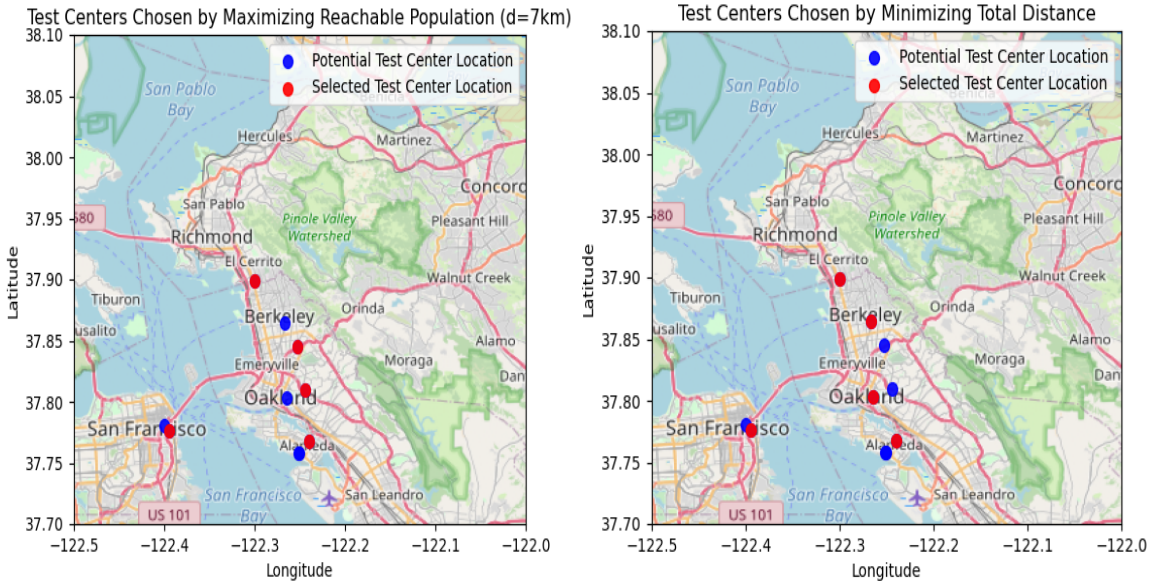
Number of test centers/*d*(km)	3	6	9	12	15	18
10	0.46	0.48	0.49	0.49	0.5	0.49
20	0.99	0.99	1	1	1.01	1.01
30	104.53	1.71	1.56	1.59	1.59	1.58
40	TLE	2.51	2.17	2.22	2.21	2.14
50	TLE	6.88	2.8	2.9	2.88	2.81
60	TLE	49.3	3.48	3.62	3.68	3.49
70	TLE	TLE	4.15	4.42	4.47	4.21
80	TLE	TLE	5	5.28	5.41	4.97
90	TLE	414.2	5.81	6.3	6.33	5.81
100	TLE	TLE	6.7	7.15	7.24	6.66
106	TLE	TLE	7.05	7.65	7.74	7.07

**TLE = TIME LIMIT EXCEEDED*

3.3 Results

While both algorithms chose a different solution for both cities, there are still commonalities between the two results. For starters, both the first and second algorithm chose test centers 2, 3, and 4 in Berkley. This suggests that these three test centers are of very high value. As for Ghent, both models also had some overlapping choices. Similar

to Berkeley, these places are likely good choices to open a test center. The maps below show the test centers chosen in Berkeley.

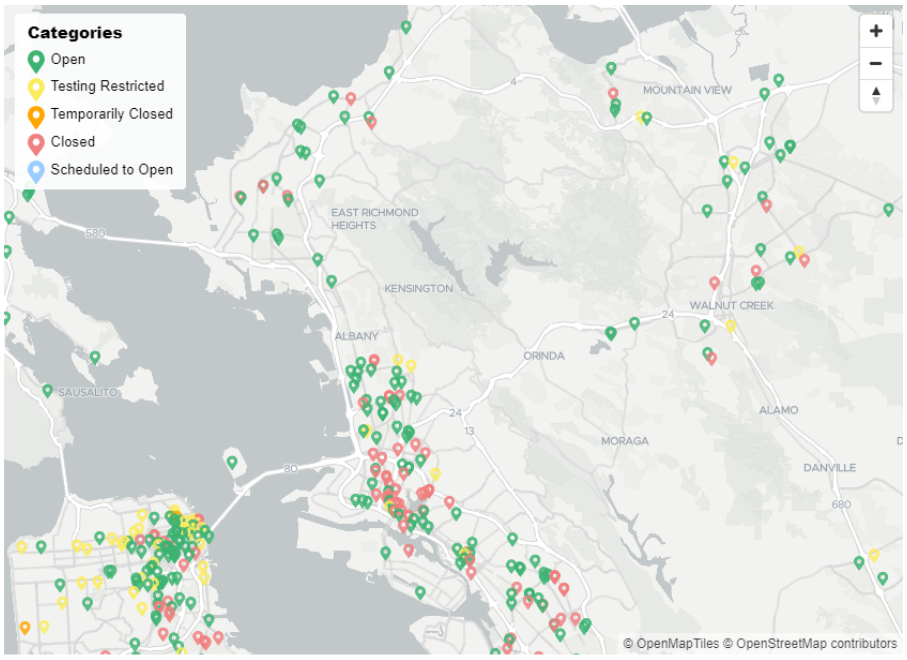


Based on the results, it seems the upper bound of the number of clusters for the first algorithm situates somewhere between 1,100 to 1,300, assuming a maximum time of ten minutes is given on the testing machine. However, it is not safe to assume that the number of clusters is decisive enough to determine the running speed, as other variables such as the number of potential test centers itself also seem to have an effect on the running time. To reduce the first algorithm's run time, it can be paired up with the second. The second algorithm can first be used to reduce the number of test centers, then the first algorithm may be used to minimize total traveling distance.

4. Discussion and Conclusion

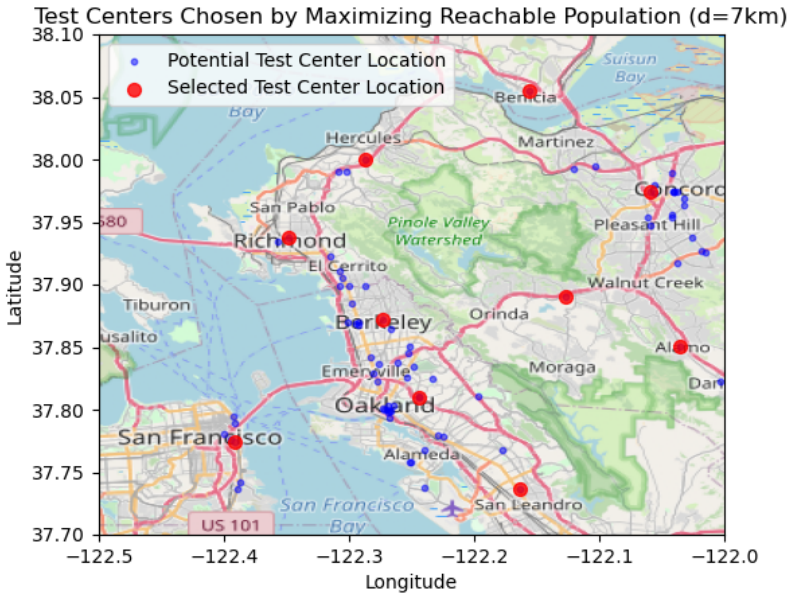
4.1 Applicability

While the two models built in this study do not reflect their respective regions accurately enough, there are still striking similarities between real-life decisions and the decisions made by the algorithms. Since the first algorithm could not read the full data, the map below details a comparison between the results from the second algorithm and a real map.



Source: URISA's GISCorps • Last updated August 3rd, 2021

Location of Test Centers in the San Francisco Bay Area, a screenshot from abc7news.com (Martichoux and Feingold).



Ten locations were chosen out of 79 (x = 10)

The locations for the test centers selected by the second algorithm are quite spread out, yet this algorithm also follows the pattern of concentrating in areas of denser population, similar to how real test centers are located. This shows that the second algorithm is definitely applicable in real life, and would produce reasonable results. Something worth noting however, is that both algorithms do not take into account many factors that would influence real-life decisions, such as the feasibility of opening a test

center, the severity of COVID-19 in different locations, the actual driving distances, and many more.

As for actual emergencies, where hospitals or other facilities must be opened as fast as possible, this tool can be applied as long as accurate data is fed. While there might be concerns regarding the first algorithm's computing speed, it is rarely affected by the number of buildings. This means that the sizes of the cities will not have many limitations. Therefore, as long as there is a controlled number of potential locations to open up a certain facility, the first algorithm can run with more acceptable times. In the end, although this project did not apply the two algorithms in a scenario close enough to real life, these two algorithms can still serve as a way to create suggestions of the general areas best suited for new facilities.

4.2 Implications for Further Research

In the future, to improve on the current algorithms, three general directions can be taken. First, these algorithms may be implemented on a different solver than *vZ*. Using solvers that take heuristic approaches may make it possible to compute a result much faster than *vZ*. However, the tradeoff would be the accuracy. If the user seeks accuracy over computing speed, *vZ* is still the best option since it guarantees the optimal solution. Second, the houses were preprocessed in this project but the test centers were not. The test centers could also use some preprocessing to be reduced in numbers. Since the number of potential test centers is what drives up both algorithms' computing time, decreasing the number of test centers will be very effective. Last, many of the clusters created in this project have high similarities when compared to others. In the first algorithms, houses with only one difference in ranking out of any number of test centers will be put into different clusters. In the Berkeley model, even if one ranking for the 79 test centers is different, another cluster will be made. However, most houses will likely not be forced to go to the last few test centers in their ranking list. This opens up possibilities for merging clusters to reduce computing time.

4.3 Closing Remarks

This study builds on previous works in that it uses computer-aided solutions to provide two different methodologies for similar end goals in spatial planning. The two methodologies are then used to determine where to install COVID-19 test centers. Not only do the results affirm the applicability of the algorithms, they also show that real life test centers are placed quite optimally. As the results of the second algorithm show overlap with real-life locations of test centers, they suggest that real-life test centers have high levels of accessibility. While the problem of installing COVID-19 test centers served as the main focus, this research can be applied in a variety of other ways as well. Any spatial planning problem that seeks to maximize accessibility from its citizens while installing a limited number of new facilities may use this tool. Although this study was not able to use data accurate enough to test out the two algorithms, users may feed custom data to them to produce realistic results, making it applicable in real-life spatial planning.

5. References

"About OpenStreetMap - OpenStreetMap Wiki." *Openstreetmap.org*, 2017, wiki.openstreetmap.org/wiki/About_OpenStreetMap. Accessed 11 Aug. 2021.

Bjørner, Nikolaj, et al. *NZ - an Optimizing SMT Solver*. Apr. 2015, pp. 194–199.

Boscoe, Francis P., et al. "A Nationwide Comparison of Driving Distance versus Straight-Line Distance to Hospitals." *The Professional Geographer*, vol. 64, no. 2, May 2012, pp. 188–196. *NCBI*, www.ncbi.nlm.nih.gov/pmc/articles/PMC3835347/, 10.1080/00330124.2011.583586. Accessed 14 Aug. 2021.

Buhat, Christian Alvin H., et al. "Optimal Allocation of COVID-19 Test Kits among Accredited Testing Centers in the Philippines." *Journal of Healthcare Informatics Research*, vol. 5, no. 1, 9 Nov. 2020, pp. 54–69. *Springer Link*, link.springer.com/article/10.1007/s41666-020-00081-5, 10.1007/s41666-020-00081-5. Accessed 13 Aug. 2021.

Chakrabarty, B.K. "Computer-Aided Design in Urban Development and Management—a Software for Integrated Planning and Design by Optimization." *Building and Environment*, vol. 42, no. 1, Jan. 2007, pp. 473–494, www.sciencedirect.com/science/article/abs/pii/S036013230500332X, 10.1016/j.buildenv.2005.08.010. Accessed 13 Aug. 2021.

de Moura, Leonardo, and Nikolaj Bjørner. "Z3: An Efficient SMT Solver." *Tools and Algorithms for the Construction and Analysis of Systems*, Apr. 2008, pp. 337–340, www.researchgate.net/publication/225142568_Z3_an_efficient_SMT_solver. Accessed 11 Aug. 2021.

Kuhlman, Dave. "A Python Book: Beginning Python, Advanced Python, and Python Exercises." *Www.davekuhlman.org*, 5 Aug. 2014, www.davekuhlman.org/python_book_01.html. Accessed 30 Aug. 2021.

Martichoux, Alix, and Lindsey Feingold. "Map Shows Everywhere You Can Get a COVID-19 Test in the Bay Area." *ABC7 San Francisco*, ABC News, 19 Nov. 2020, abc7news.com/covid-testing-near-me-san-francisco-test-free/8098382/. Accessed 19 Aug. 2021.

Mitchell, William J. *Computer-Aided Architectural Design*. New York, Van Nostrand Reinhold, 1979.

Sarubbi, Joao F. M, et al. "A Strategy for Clustering Students Minimizing the Number of Bus Stops for Solving the School Bus Routing Problem." *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2016, pp. 1175–1180, ieeexplore.ieee.org/document/7502983. Accessed 16 Aug. 2021.

World Health Organization. "WHO COVID-19 Dashboard." *Covid19.Who.int*, World Health Organization, 2021, covid19.who.int/.

6. Appendix

6.1 Additional Data

Number of test centers/*d*(km)	3	6	9	12	15	18
10	40.17	58.72	73.44	81.11	84.62	88.31
20	46.24	73.88	90.18	96.67	98.5	99.7
30	TLE	78.47	93.94	97.56	99.06	99.81
40	TLE	79.31	94.75	97.79	99.06	99.81
50	TLE	89.77	98.64	99.87	100	100
60	TLE	91.33	98.85	99.87	100	100
70	TLE	TLE	98.96	99.87	100	100
79	TLE	TLE	99.23	99.91	100	100

*Percentage of population satisfied using the second algorithm in the Berkeley model as number of test centers and d changes; *TLE = TIME LIMIT EXCEEDED*

Number of test centers/*d*(km)	3	6	9	12	15	18
10	66.48	88.01	96.75	99.97	100	100
20	66.48	92.38	98.24	100	100	100
30	70.02	96.66	99.94	100	100	100
40	TLE	96.66	100	100	100	100
50	TLE	96.74	100	100	100	100
60	TLE	97.42	100	100	100	100
70	TLE	TLE	100	100	100	100
80	TLE	TLE	100	100	100	100
90	TLE	TLE	100	100	100	100
100	TLE	98	100	100	100	100
106	TLE	TLE	100	100	100	100

*Percentage of population satisfied using the second algorithm in the Ghent model as number of test centers and d changes; *TLE = TIME LIMIT EXCEEDED*

Number of test centers	Preprocessing Time (s)	Solving Time (s)	Total Time (s)
10	0.46	0.02	0.48
20	0.92	0.07	0.99
30	1.38	0.33	1.71
40	1.8	0.71	2.51
50	2.26	4.63	6.88
60	2.71	46.59	49.3
70	3.22	TLE	TLE
80	3.73	TLE	TLE
90	4	410.21	414.2
100	4.68	TLE	TLE
106	4.97	TLE	TLE

*Running time of the second algorithm using the Ghent model (d = 6 km); *TLE = TIME LIMIT EXCEEDED*

Number of test centers/*d*(km)	3	6	9	12	15	18
10	24	52	75	76	67	49
20	102	155	231	216	192	140
30	192	310	459	441	386	281
40	346	509	702	675	584	422
50	516	709	952	875	787	574
60	665	920	1163	1116	1022	717
70	895	1143	1365	1341	1220	847
80	1145	1421	1622	1645	1462	952
90	1382	1700	1913	1940	1726	1119
100	1601	1902	2121	2134	1908	1265
106	1766	1996	2195	2207	1976	1285

*Number of clusters formed using the second algorithm in the Ghent model as number of test centers and d changes; *Shading = TIME LIMIT EXCEEDED*

6.2 Additional Information

The machine which ran the tests are on the following specs:

1. CPU: Intel (R) Core (TM) i7-9700K CPU @ 3.60GHz
2. RAM: 16GB, 2666 Mhz

The link to the code in the project:

https://github.com/AaaaronH/Yichen_Aaron_Huang_COVID-19_Research_Paper_Code