SSC19-WP1-05

Optimized Ground Station Placement for a Mega Constellation using a Genetic Algorithm

Joseph Kopacz, Jason Roney, Roman Herschitz
University of Denver
2199 S University Blvd, Denver, CO; 720-472-6606
Joseph.Kopacz@du.edu

ABSTRACT

This paper will review results and discuss a novel method to address the multiple-objective optimization problem of ground station placement; enabling continuous communication with the mega constellation defined in Optimized Continuous Global Coverage Constellation using a Genetic Algorithm. A genetic algorithm implemented in MATLAB explored the globe utilizing Satellite Tool Kit to determining the optimal number of ground stations and their placement – considering local infrastructure available and the constellation connectivity during a 24-hour period. A new revenue-based fitness function evaluated these parameters and the potential revenue to determine the most profitable configuration.

INTRODUCTION

New mega constellations have been proposed to provide continuous global coverage constellations, but require an unprecedent network of ground support. These constellations range from 300-3000 small satellites. This paper will attempt to address the multiple-objective optimization problem of ground station support for a mega communications constellation in a Low Earth Orbit (LEO). This optimization will utilize a genetic algorithm using a variable length chromosome.

METHOD

STK Link

A constellations connectivity is dependent on the spacecraft position in orbit relative to the fixed ground station location on the globe. For GEO spacecraft this is simple because they are continuously in view of a single ground station. For LEO constellations, the ground stations in view can change rapidly, and thus a large network of ground stations is required to keep LEO constellations active. System Tool Kit (STK) [1] can generate simulations of massive constellations and the ground stations to keep the spacecraft connected with ease; however, STK lacks the flexibility to perform high level logic-based modifications of the ground station network. The software developer kit available in STK allows users to interface with STK via C#, MATLAB, Java, and Python. This section will cover the implementation of MATLAB [2] to generate the ground network and evaluate the constellations connectivity.

The MATLAB function created for this study that creates the STK simulation accepts 3 inputs: the number of facilities, and a list of longitudes and latitudes. Initially, an instance of STK is created, and a simulation is loaded. Early experimentation showed it was orders of magnitude faster to load a simulation with thousands of spacecraft than it was to generate a simulation from scratch. The constellation loaded is the same constellation found in *Optimized Continuous Global Coverage Constellation using a Genetic Algorithm;* however, the number of spacecraft has been decreased to 190 to simplify the problem. This greatly reduces the simulation time, and since the satellites are evenly spaced along the planes it gives a good reference of the percentage of spacecraft that are covered at any one time. The parameters used to generate the constellation can be found in Table 1. The graphical representation of the simplified constellation can be seen in Figure 1, with the un-simplified constellation found in Figure 2.

Table 1: Optimal LEO Constellation

Variable	Optimal
Altitude (Km)	1396
Inclination (deg)	91
Number of Planes	19
Number of Sats (Total)	665 – (simplified to 190)
RAAN (deg)	275

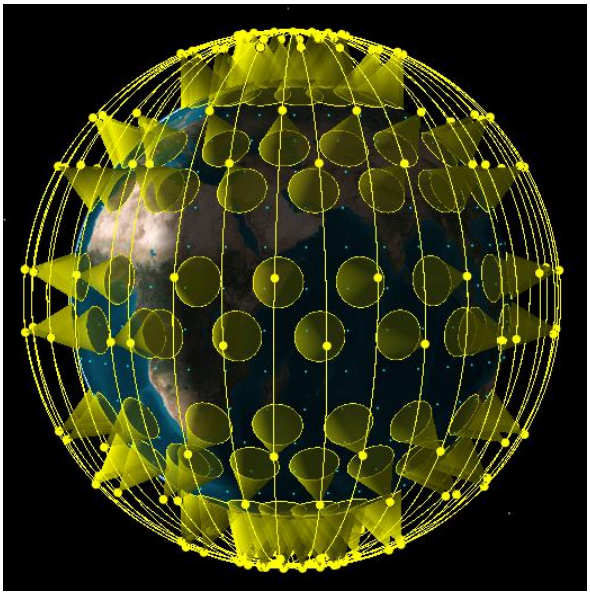

Figure 1: Simplified Optimal LEO Constellation

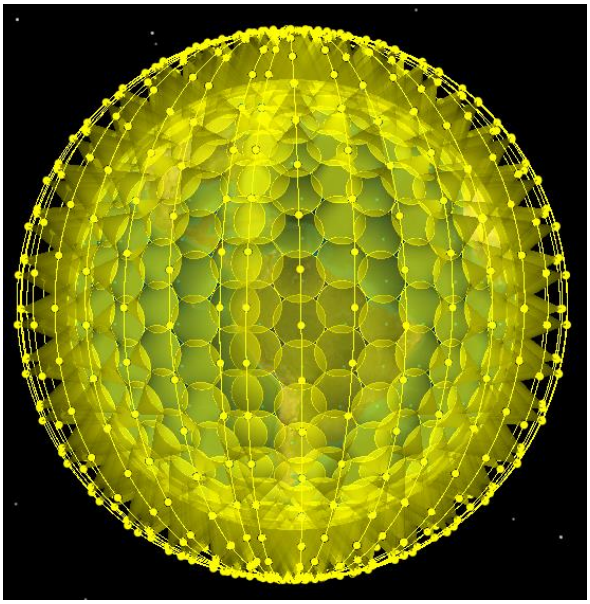

Figure 2: Un-Simplified Optimal LEO Constellation

Once the constellation has been loaded, the MATLAB function commands STK to place a ground station at the specified latitude and longitude, looping through each ground station until all are placed in the simulation. Each satellite and ground station will be called by name in a nested loop, so MATLAB generates the names for all the assets. It should be noted that the satellite names were defined in the loaded simulation, which followed the naming convention Test_SatPlane#Satellite# — i.e. Test_Sat0301 is the first satellite in plane 3. The ground stations follow the naming convention MyFacility# – i.e. MyFacility1. Once the script has generated the names of all the ground stations and satellites, the access intervals between each spacecraft and each ground station can be requested – an example of the data returned can be seen in Figure 3. After all the access intervals are returned from STK a parallel loop is initiated in MATLAB. This loop runs through every satellite and ground station combo and determines if a satellite is connected to a ground station during 10 instantaneous periods that were linearly spaced over the 24-hour period of the simulation. The average connectivity of all the spacecraft is finally calculated. A graphical representation of a satellite's connectivity during the 24-hour period can be seen in Figure 4. In this figure the ground track of Test_Sat0101 can be seen. The satellite is just about to enter the coverage of MyFacility01. The blue lines show the ground track of the satellite during the coverage periods of MyFacility01. The coverage periods of MyFacility20 can also be seen in pink.

Finally, the revenue of the connected satellites can be calculated. The average connectivity of each satellite over a 24 hour period is multiplied by the approximate five-year revenue of the constellation, then the cost of a ground station is multiplied by the number of ground stations and is subtracted. Each ground station was estimated at $1.5M. The actual cost of the ground stations is irrelevant as long as it is consistent across simulations. We are simply looking to reward the simulation for good coverage and degrade the fitness for greater number of ground stations.

Figure 5 and Figure 6 show that the satellite connectivity begins once the satellite has broken the 90-degree half angle barrier. Essentially, once a satellite comes around the globe into view – assuming no global topology – it begins connectivity. It is possible to increase the real-world accuracy of the simulation by using the half angle defined by the actual communications hardware being used. Additionally, it is possible to have STK simulate the local topography

MyFacility 20-To-Test_Sat0101			
Access	Start Time (UTCG)	Stop Time (UTCG)	Duration (sec)
1	Oct 5 2016 19:41:54.178	Oct 5 2016 20:02:20.125	1225.947
2	Oct 5 2016 21:34:56.347	Oct 5 2016 21:55:12.590	1216.243
3	Oct 6 2016 07:51:45.027	Oct 6 2016 08:12:51.140	1266.114
4	Oct 6 2016 09:45:25.883	Oct 6 2016 10:04:45.949	1161.066

Figure 3: Access Times Facility 20 to Satellite 0101

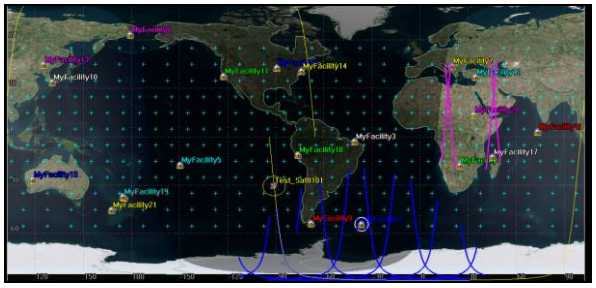

Figure 4: Graphical Access Times from Ground Station 1 and 20 to Satellite 0101

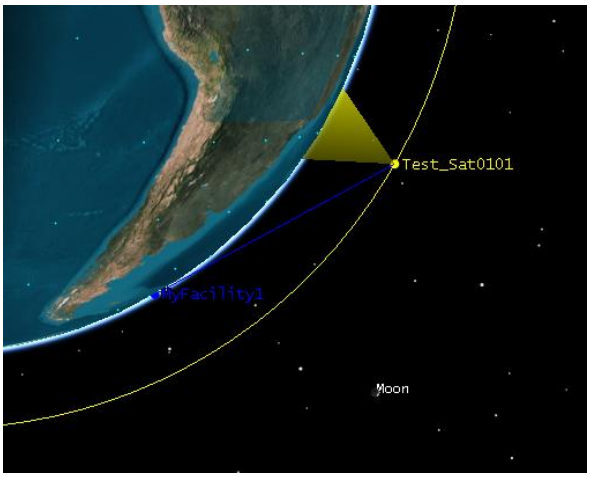

Figure 5: Connectivity Established at 90 deg Half Angle (3D)

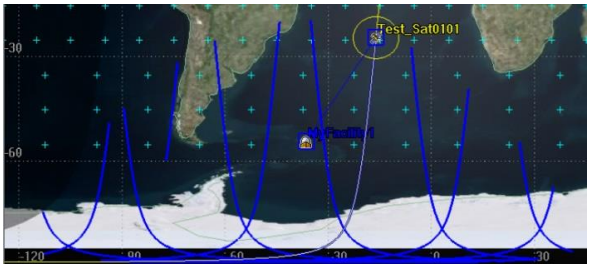

Figure 6: Connectivity Established at 90 deg Half Angle (2D)

MATLAB Genetic Algorithm

A genetic algorithm was selected to optimize this non-linear, discontinuous, multi-objective problem, due to its ability to find a global solution where many other methods fail. Genetic algorithms are a metaheuristic process similar to natural selection. At a high-level, the genetic algorithms select the top performers of a population then mutates chromosomes, performs breeding, and finally computes new fitness scores. This continues for several generations until a single set of optimized parameters is found. This implementation of a genetic algorithm does not use cross-over, but instead mutates the top 10% of each population.

The genetic algorithm creates an initial population by randomly selecting the number of facilities and then randomly selecting where the facilities will be placed. This random process creates a variable length chromosome which adds immense complexity to the genetic algorithm by not only optimizing how many ground stations are needed, but also optimizing where those ground stations should be placed. Cities as opposed to any location on the globe were selected as they have the power and network connectivity required for ground stations. The initial population is then fed into STK where the coverage of each chromosome is evaluated and returned to MATLAB where the fitness score can be calculated.

The simplified list of cities was generated from the board game Pandemic [4], which provides an assorted set of major cities with great geographical diversity. GPS coordinates (lat-lon sets) for these cities were found and tabulated for the algorithm to reference.

The initial population is created and sorted based on its fitness score and the top 10% of samples are selected as parents. A parent is then randomly selected for mutation, and the number of cities is changed. If the number of cities is smaller than the current list in the parent, random cities are deleted. If the number of cities is greater than the current list, new cities are randomly selected and added to the list, assuming the selected city is not already in the parents list. The final case is infrequently chosen but occurs if the randomly selected number of cities is equal to the size of the current list. In this case a few random cities are chosen to be replaced. This process is repeated until an entire population is created. The calculate_coverage function is then called for each newly mutated sample. This function requests the coverage time for all facilities and satellite combinations from STK and returns them to MATLAB where a custom script was created to manage the large volume of information in parallel. The fitness and coverage scores are then calculated and returned. This calculation is repeated until the maximum number of generations is reached.

During initial testing it was found that the ground station optimization problem would not converge without a forcing function. To help solution convergence, a linearly decaying selection was implemented. Initially the algorithm has significant range in how much it can mutate the selected

chromosomes in terms of the number of cities, but toward the final generations it is only allowed to change a few cities. Equation 1 and Equation 2 are used to determine the allowed range of mutation used in the creation of the next generation's chromosomes. Figure 7 has been added to show how the selectable mutation range linearly converges from the initial maximum and minimum values towards the previous generations optimal value. In this figure the previous optimal value was made static at 14 to show how the allowable range collapses. The final range collapses to 10% of the initial range, or 5% on either side of the latest optimal value. Additional protections were added to cover edge cases ensuring the range never exceeded the max or min allowed values. Figure 8 shows how the range collapses when using a randomized center value within the range.

Equation 1: Max Value

$$\vee mutation = floor(previous + ((1 - (\frac{gen}{\vee gen})) * (\vee allowed - previous)) + ceil(\frac{1}{2} * (\vee allowed - \wedge allowed)))$$

Equation 2: Min Value

$$\wedge mutation = ceil(previous + ((1 - (\frac{gen}{\vee gen})) * (previous - \wedge allowed)) - ceil(\frac{1}{2} * (\vee allowed - \wedge allowed)))$$

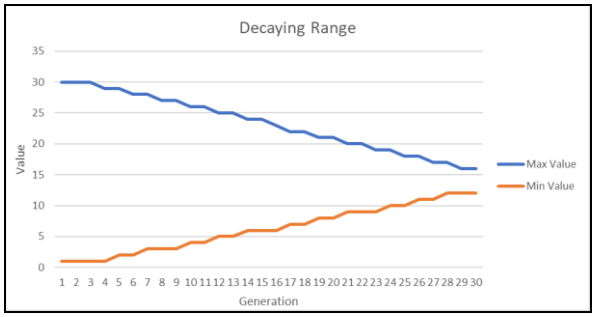

Figure 7: Decaying Range

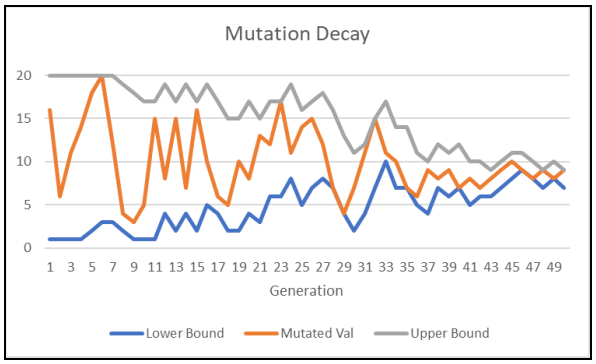

Figure 8: Decaying Range with Random Center

Due to the amount of time to simulate a single city set, a verification method is required to determine if this location permutation had been simulated before. Because the city permutation could potentially be in a different order each time a hashing function was selected. The hashing function was useful when using the reduced city set, since it is easier to select the same run, but was found to be unnecessary when using the larger city list. All facility Latitudes and Longitudes were independently sent to the hashing function found in Equation 3, and stored in a database of previous runs. In hindsight it would have been easier to assign an ID to each city and use this value in the hashing equation instead of computing the hash for each lat-lon pair.

Equation 3: Hashing [3]

$$(\prod_{0}^{N} Facility_N * 2 + 1779033703)/2$$

RESULTS

Simplified City List with Decay

Simulations allowing for placement at any point on the globe would not converge in a reasonable amount of time, so it was decided to reduce the scope of the problem. A list of 48 major cities was generated using the board game Pandemic [4]. The game board seen in Figure 9 shows a good distribution of major cities across the entire globe. This list is missing valuable cities in higher latitudes and across central Asia but provided a good starting point. Even using this simplified list of cities there are still 2.7 * 10^{14} potential combinations – assuming 1 to 30 cities is chosen out of the list of 48 cities.

A simulation was run for 50 generations with a population of 15 chromosomes in each generation. This simulation had a small population size for general genetic algorithms, but it was hoped that the greater

number of generations and more regular down selection would help the problem converge faster. It is unclear if this small population size helped the problem converge as other simulations, which used much larger populations, also failed to converge without the forcing function found in Equation 1 and Equation 2. Figure 10, shows the max fitness score found in each generation. As expected, the initial populations score is very low compared to the remainder of the generations. Small increases are found in the first 2 generations, but a large spike is seen in generation 3. Data examination shows that the simulation actively explores a variety of locations and number of locations to randomly find an acceptable solution set. Lots of variability is seen in the following generations, but the simulation starts to slowly converge between generations 25 and 35. The optimum solution is found in generation 48 after several more generations with slight variability. Figure 11 shows the average fitness with the maximum and minimum fitness of each generation. In later generations, the tightening bands show there is much less variability in the simulation. This tightening is due to the linearly decaying algorithm that was implemented to force less variability and eventually convergence in later generations. In generation 16 the fitness diversity is greatly diminished; inspection of the data shows that this generation only used high numbers of ground stations, specifically only between 10 and 30. All solutions found in this generation were relatively fit. In later generations the average is near the top of the diversity band as the average fitness score is very high, again showing the diversity is greatly diminished as the decaying search range collapses.

Table 2 shows the convergence of the solution. Eventually selecting 12 ground stations, achieving 68% coverage of the constellation. Additionally, this table shows how the simulation initially selected 9 ground stations as the optimal solution, and quickly ramped up 21 ground stations before minimizing the number of ground stations.

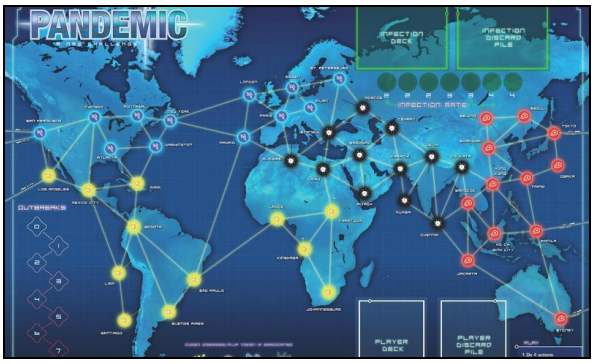

Figure 9: Pandemic Game Board [4]

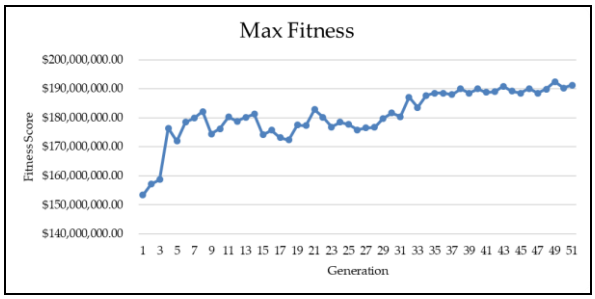

Figure 10: Max Fitness (Simplified City List)

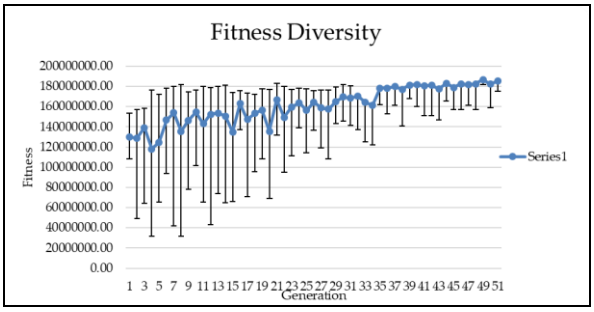

Figure 11: Fitness Diversity (Simplified City List)

Table 2: Generational Max Fitness and Number of Ground Stations (Simplified City List)

Generation	Max Fitness	Coverage At Max Fitness	Number of Ground Stations
0	$ 153,384,047.30	0.54	9
10	$ 180,283,725.94	0.68	20
20	$ 182,870,554.77	0.66	15
30	$ 180,421,449.66	0.68	21
40	$ 188,792,408.82	0.67	14
48	$ 192,447,498.31	0.68	12
50	$ 191,275,043.06	0.68	13

Figure 12 shows the optimal twelve ground stations found in generation 48 – these cities and their coordinates can be found in Table 3. A 53-degree sensor was placed on each ground station to visually show the ground track each ground station can cover – the 53-degree sensor angle was found through trial and error comparing the satellite ground path and the coverage area. This figure shows excellent placement of the ground stations with minimal overlap of the coverage, illustrating the algorithm is correctly trying to maximize constellation connectivity, while also minimizing the number of ground stations. Facility 1 in Moscow, Russia turns out to be an extremely important station as it is responsible for coverage of the North

Pole, Europe, and much of Russia. Its importance as a ground station is further exemplified by Figure 13 which shows that due to the polar orbits a huge number of satellites can be found at the poles at any one time. Additionally, in this solution most of the land masses have coverage with coverage only missing over the South Pole and some of the oceans. Visual inspection shows a ground station in Hawaii and Natal, Brazil would help achieve more complete coverage of the oceans. Hawaii was an option in the cities list, but not chosen in the final optimized solution, potentially because the cost of the additional ground station, as expressed in the fitness function, did not justify the coverage. However, it may have simply been left off the list because it wasn't randomly chosen in the last several generations.

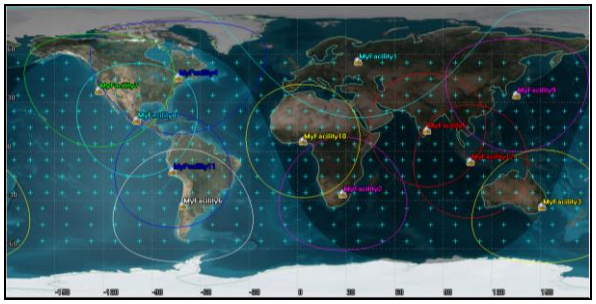

Figure 12: Optimal Ground Station Placement (Simplified City List)

Table 3: Optimal Ground Station Placement (Simplified City List)

Lat	Lon	City	Country
55.7522	37.6155	Moscow	Russia
-26.17	28.03	Johannesburg	South Africa
-33.92	151.1852	Sydney	Australia
45.5	-73.5833	Montreal	Canada
13.09	80.28	Chennai	India
-33.4489	-70.6693	Santiago	Chile
37.7749	-122.419	San Francisco	USA
19.4424	-99.131	Mexico City	Mexico
34.75	135.4601	Osaka	Japan
6.4433	3.3915	Lagos	Nigeria
-12.048	-77.0501	Lima	Peru
-6.1744	106.8294	Jakarta	Indonesia

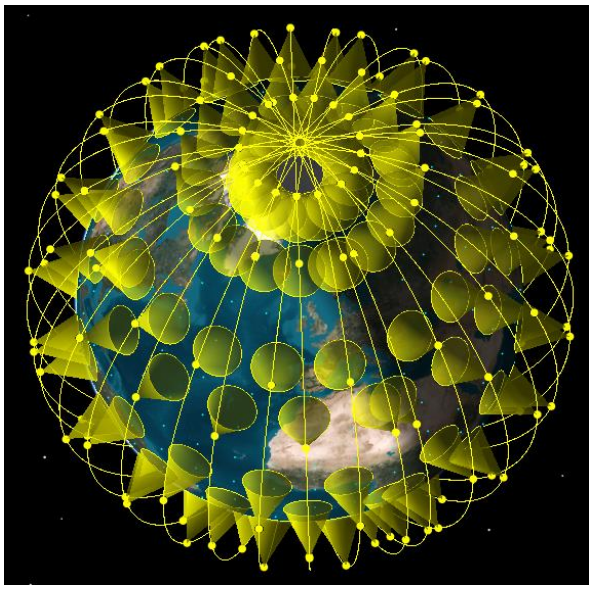

Figure 13: Satellite Pole Coverage

Large City List with Decay

After positive results were found with the reduced city list a larger list of cities was used to determine if constellation coverage could be improved. A database of nearly 13,000 cities and their coordinates was provided by SimpleMaps [5]. This list was generated utilizing lists from the National Geospatial Intelligence Agency, the US Census Bureau, the US Geological Survey, and NASA. A plot of the available cities can be seen in Figure 14. This list greatly increases the potential combinations to $7.49 * 10^{90}$ – assuming 1 to 30 cities is selected out of 12,894.

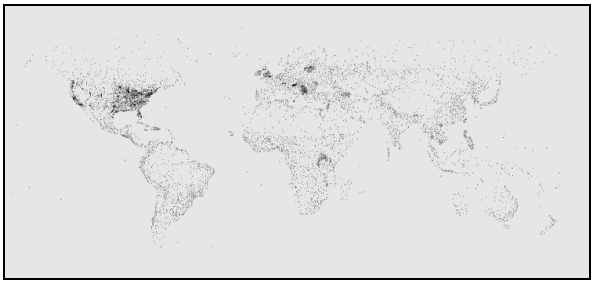

Figure 14: Available Cities in SimpleMaps Basic Database [5]

Again, 50 generations of a 15 chromosome population were run. Figure 15 shows a fairly linear climb in fitness score all the way to generation 29. At which point the fitness score continues to climb, but irregularly and less quickly. Like before it is assumed that more regular feedback using a lower number of chromosomes and a greater number of generations had an impact on how quickly the solution converges;

however, this is unconfirmed as other simulations that used a large number of chromosomes and fewer generations converge no slower. The fitness diversity given in Figure 16 shows good genetic diversity which rapidly collapsed due to the linear decay function that was implemented. Again, in the final generations the average stays near the top of the band. Table 4 supports these results and shows the fitness score increasing all the way to generation 50. This table shows that within the first 20 generation the simulation initially explored optimizing coverage with a greater number of ground stations, but then started to minimize the number exploring between 18-21 ground stations in the final generations before landing on 21.

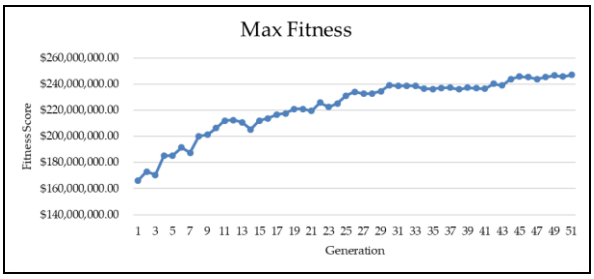

Figure 15: Max Fitness (Large City List)

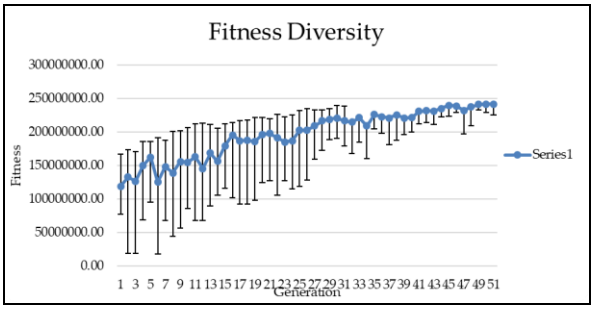

Figure 16: Fitness Diversity (Large City List)

Table 4: Generational Max Fitness and Number of Ground Stations (Large City List)

Generation	Max Fitness	Coverage At Max Fitness	Number of Ground Stations
0	$ 166,165,773	0.61	15
1	$ 173,085,222	0.68	26
5	$ 191,375,630	0.75	28
10	$ 212,003,469	0.79	22
15	$ 213,752,868	0.80	24
20	$ 219,451,372	0.80	19
25	$ 234,027,113	0.84	19
30	$ 238,501,064	0.85	17
35	$ 236,975,015	0.85	19
40	$ 236,293,877	0.86	20
45	$ 245,301,358	0.88	20
46	$ 243,801,358	0.88	21
47	$ 245,301,358	0.88	20
48	$ 246,663,634	0.88	18
49	$ 245,792,675	0.89	20
50	$ 247,076,805	0.90	21

Figure 17 shows the optimal ground station placement for the 21 ground stations determined in generation 50. These location names and their coordinates can be found in Table 5. This chart illustrates the 90% constellation coverage, but also shows significant overlap. The bottom left corner specifically shows 2 ground stations nearly on top of each other providing identical coverage, and a third ground station with significant overlap nearby. Station 6 in Beringovskiy, Russia provides large coverage for the North Pole as Moscow did in the previous simulations.

A shortcoming found in the simplified cities solution list was lack of coverage in the ocean west of South America. The larger city solution list has Papeete, French Polynesia as a location which rectifies this concern. This location would have been a major benefit in the simplified cities list solution.

Many of the names in the list were not instantly recognizable so these locations were further investigated for ground station viability. Figure 18 is a Google Maps [6] street view of Grytviken in South Georgia. It turns out this city is only inhabited during the summer months, which would not immediately exclude it from a ground station location; however, the

city appears to lack the basic infrastructure such as power, and would likely require costly power generation and a data connection. SimpleMaps [5] has a paid list with a city ranking based on cities importance. The ranking criteria are proprietary; however, it is likely based on a city's available infrastructure and population. A future study could include the importance ranking in the fitness function. A way of doing this would be to assume a less important city would increase the cost of developing a ground station as these cities would have less immediately available infrastructure and support.

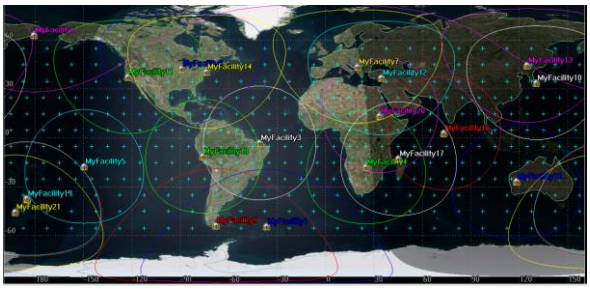

Figure 17: Optimal Ground Station Placement (Large City List)

Table 5: Optimal Ground Station Placement (Large City List)

Lat	Lon	City	Country
-54.2806	-36.508	Grytviken	South Georgia And South Sandwich Islands
-37.7783	175.2896	Hamilton	New Zealand
-3.1195	-40.84	Granja	Brazil
-17.8096	25.15	Kasane	Botswana
-17.5334	-149.567	Papeete	French Polynesia
63.0655	179.3067	Beringovskiy	Russia
43.8582	19.8441	Uzice	Serbia
42.45	-89.0631	Rockton	United States
-53.7914	-67.699	Rio Grande	Argentina
33.8704	130.82	Kitakyushu	Japan
37.586	-122.367	Burlingame	United States
36.8004	34.6128	Mersin	Turkey
44.4304	125.1701	Nongan	China
40.6746	-73.6721	Malverne	United States
-26.6	118.4833	Meekatharra	Australia
2.9217	73.5811	Muli	Maldives
-12.7871	45.275	Dzaoudzi	Mayotte
-11.1496	-76.01	Junin	Peru
-37.2015	174.9033	Pukekohe	New Zealand
13.55	33.6	Sennar	Sudan
-45.4074	167.7585	Te Anau	New Zealand

Figure 18: Street View Grytviken South Georgia and South Sandwich Islands [6]

CONCLUSIONS

Even though the scope of the problem is complex due to the number of permutations in the ground station optimization problem, a genetic algorithm can be used to successfully find a solution as shown in this paper. Two possible solutions were shown in the results. First, using the reduced cities list, the algorithm was able to find excellent coverage and minimal overlap. Visual inspection of the optimized locations immediately showed locations that could increase coverage over the oceans. This result led to the insight that the algorithm could find an even better solution given more time and more options. By expanding to the large city list the algorithm was able to find coverage where previously there was none, but the list was so large that only an infinitesimal piece of the total design space was explored. Using the simplified city list 68% of the constellation was covered using 13 ground stations. In the large simulation 90% coverage was found using 21 ground stations, but significant overlap was seen.

As it stands the large city list simulation is exploring 750 solutions out of $7.4 * 10^{90}$. Increasing the speed of the simulation would allow much more of the design space to be explored; however, there are other ways to optimize the solutions.

One way would be to utilize the city rankings based on qualities such as importance. One could start by only using tier 1 and 2 cities which would greatly reduce the number of cities, but still provide a bigger list than those used in this paper's simplified list. Reviewing the plot of available cities seen in Figure 14 also illuminates a potential to prune the cities list based on density. The plot shows that the densest cities are in the US and Europe. All these cities are not required for constellation coverage, so an algorithm could be developed to simplify the list based on relative distance. Additionally, an intelligent deletion method could be used to prune ground stations that are densely packed.

Another thought would be to implement cross-over. It is possible that allowing individual chromosomes to develop over several generations could find mini-

optimal solutions; for example, one could implement the algorithm to provide optimal coverage of the southern hemisphere, and then breed those solutions back into the larger group.

REFERENCES

[1] Analytical Graphics, Inc., *Systems Tool Kit*, Exton, Pennsylvania, 2018.

[2] The MathWorks, Inc., *MATLAB and Statistics Toolbox Release 2018b,* Natick, Massachusetts, 2018.

[3] abc, "Good has function for permutations," 8 Oct 2009. [Online]. Available: https://stackoverflow.com/questions/1536393/good-hash-function-for-permutations. [Accessed Jan 2019].

[4] M. Leacock, *PANDEMIC,* Z-Man Games, 2008.

[5] SimpleMaps, "World Cities Database," [Online]. Available: https://simplemaps.com/data/world-cities. [Accessed January 2019].

[6] Alphabet, "Google Maps," March 2014. [Online]. Available: https://www.google.com/maps/@-54.2813701,-36.5097297,2a,75y,68.85h,89.06t/data=!3m6!1e1!3m4!1sD-MkO4xJMK92VcvQCAewqw!2e0!7i13312!8i6656. [Accessed February 2019].