P-Median Problem

Jarow Myers
Evolutionary Computation

Introduction

In this project, the P-Median problem was solved. In the P-Median problem, there exists a set of points on a graph, and the goal is to find a subset of size P (the median set) of these points such that the sum of the distances from each other point (the demand set) to the nearest point in the median set is minimized. The challenge that the algorithms in this project are attempting to solve is determining which points of the graph should belong in the median set as to minimize that distance. The result is a 'solution' in which ideally every point in the graph is near a point in the median set.

The graph representation used in this project is a simple list of coordinate points. The graph is essentially fully-connected, as each pair of points has an 'edge' that is the value of the Euclidean distance between them.

This project covers using a Simple Genetic Algorithm (SGA or GA), as well as using Simulated Annealing and Foolish Hill-Climbing (SA/HC), to find solutions to the P-Median problem.

Chromosome and Fitness

The chromosome design for this project is a binary string. The length of the string is equal to the size of the graph, and each allele corresponds to a specific point in the graph (i.e. allele index 5 corresponds to the point of the graph in the point list at index 5). In the bit string, an allele set to '1' means the corresponding point is in the median set. Alleles of '0' mean that point lies in the demand set. As a solution should have P median points, a chromosome should only have P 1's in its genes.

Any chromosome with more or less than P 1's is infeasible. To deal with infeasibles, chromosomes are 'fixed up' so that they have the correct number of median points and become feasible. The mutation operators (discussed later) ensure that a chromosome remains feasible, but the crossover operators do not. If two chromosomes are crossed over and the resulting children have the incorrect number of median points, they are fixed up using the following rules:

1. If a chromosome has too many 1's, randomly remove 1's until feasible.

2. If a chromosome has too few 1's after crossover, then it inherits additional 1's from the *new* parent until it becomes feasible.

Using these rules for fixing, infeasible chromosomes are no longer an issue, and fitness only has to concern itself with feasible solutions.

The fitness function used in this project is defined as follows: the fitness of a chromosome is equal to the sum (over all points in its demand set) of the distance from the point to the nearest median in the median set. This is a computationally expensive evaluation, because each point in the demand set has to loop through the median set and calculate its distance to each median to find the nearest, and then add that nearest distance to the total sum. In the end, each point in the demand set contributes to the fitness its 'distance to the median set'. The goal is to minimize this fitness, meaning that the most fit individual is the one where the total distance from each demand point to the median set is minimized.

Initial chromosomes for the starting population and initial SA/HC solution were generated randomly. This was done using random sampling to select P random indices to give a chromosome P randomly chosen medians.

Selection, Crossover, and Mutation Operators

The two selection operators in this project are Roulette and Tournament. Roulette works in the standard way. Each chromosome in the population gets an area on a roulette wheel proportional to its fitness, where higher fit (low fitness value) individuals are more likely to be selected. The wheel is spun repeatedly to create the parent pool. With Tournament selection, two chromosomes are chosen randomly, and then based on a tournament parameter, either the more fit or less fit individual gets selected for the parent pool. The parameter value used was 0.75, meaning the more fit individual wins 75% of the time, and 25% percent of the time the less fit individual gets selected. This is repeated to create the parent pool. Additionally, elitism is used, so the top two most fit parents in the population automatically get passed on to the new generation without experiencing selection, crossover, or mutation.

The two crossover operators used are Double Point and Uniform crossover. In Double Point crossover, two random split points are chosen, and then the alleles in between the points are swapped with the other parent. This creates two new children (one for each parent as a base), who each have one parent's genes outside the split points, and the other parent's genes between the split points. For Uniform crossover, a random uniform bit string is created where each character has an equal chance to be 0

or 1. Then, a child is created by selecting the base parent's allele if the corresponding character is 0, and the new parent's allele if it is 1. This creates two children, as each will use a different base parent and new parent. (Note: these crossovers may produce infeasibles, which get fixed up according to the rule described earlier.)

The two mutation operators used are referred to as Single Point and Nearest Neighbor. In Single Point, a random median point is chosen to become a demand point, and a random demand point is chosen to become a median. In this way, the median set randomly changes by one point, and the total number of medians remains constant (chromosomes stay feasible). In Nearest Neighbor, a random median point is selected, and then the demand point closest to it becomes a median, and the old median becomes a demand point. This is similar to Single Point, but attempts to use information in the graph instead of random chance to decide how the median set changes. The Nearest Neighbor algorithm is an attempt to make a small modification to the solution that is limited in its ability to drastically affect fitness, where in Single Point it is possible that the random move has a larger effect.

For Simulated Annealing and Hill Climbing, the same two mutation operators were used as the perturbation functions.

Parameters

The program asks the user to specify some of the parameters for the Genetic Algorithm, so that it can be used for a variety of parameter sets. However, for the purposes of data collection and investigation, the parameter values used are as follows:

Population Size: 100
Crossover Rate: 1 (100%)
Mutation Rate: 0.05 (5%)
Termination Criteria: 1000 generations or 100 generations where the most fit individual remains the same (elites reign for 100 generations in a row).
Tournament Rate (if used): 0.75

For Simulated Annealing and Foolish Hill-Climbing, the parameters are as follows (for Foolish, this only controls the number of iterations run):

T_0: 10
I_0: 1000
A (temp factor): 0.95
B (iteration factor): 1.02

With these parameters, SA/HC will run for a total of 71,255 iterations.

The datasets will be discussed in the next section, but the parameters dealing with them are stated here for clarity:

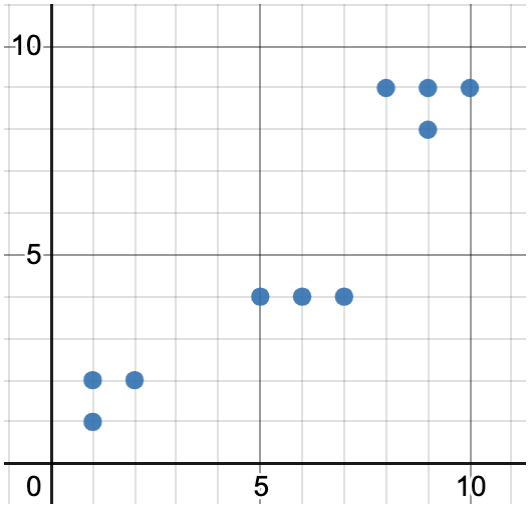
Toy Data Graph Size: 10
Toy Data P Value: 3
Medium Data Graph Size: 50
Medium Data P Value: 5
Large Data Graph Size: 100
Large Data P Value: 10

Chromosome size is equal to the graph size, which is dependent on the dataset selected.

Datasets

A total of five datasets are used in this project. They consist of a small, contrived dataset with a known optimal, a medium dataset, and three large datasets. Aside from the contrived small dataset, the datasets are randomly generated, with no known optimal.

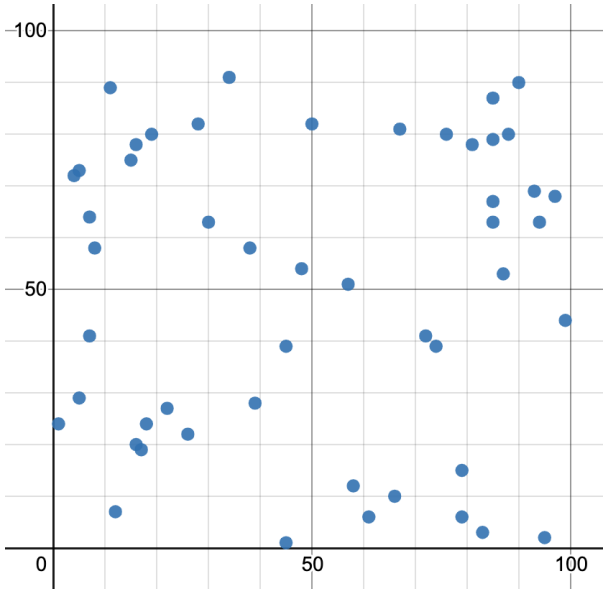
The toy contrived dataset consists of 10 points with coordinates in [0,10]. The dataset is designed to have three clusters, each with a clear centerpoint. In this way, the solution for the P-Median problem with 3 medians is clearly the three centerpoints of the clusters. The image of the data set is shown below.



Toy Dataset

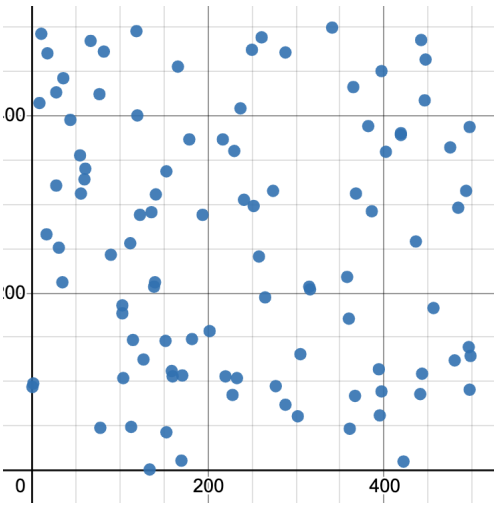
The solution to this dataset is the set of points [(1,2), (6,4), (9,9)], with a fitness of 7. Each of the algorithms was able to find the optimal solution at least once in 5 trials (if not always), proving their functionality. Further discussion on the results is in the next section.

The medium dataset consisted of 50 randomly generated points with coordinate values in [0, 100]. For data collection, this problem was attempted with a median set size of 5. No optimal solution is known. An image of the dataset is shown below.

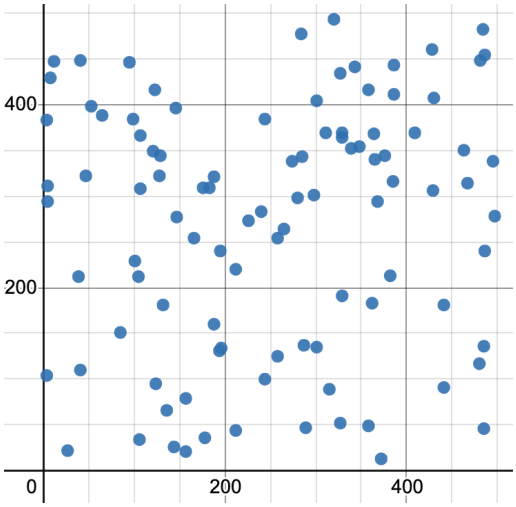


Medium Data Set

The next two large datasets consist of 100 random points with coordinates in [0, 500]. A median set size of 10 is used for the large problems. No optimal solution is known. Images of these two are below.

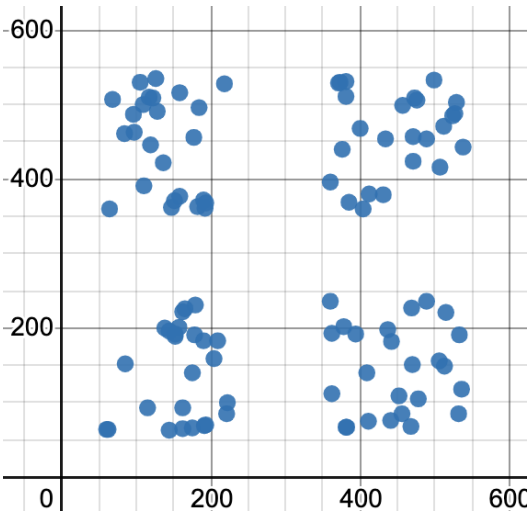


Large Dataset 1



Large Dataset 2

The third large dataset consists of 100 random points with coordinate values in [50,550]. However, in this dataset, points were generated to be in one of four even clusters. This should mean that each point in the demand set will ideally be looking for a median within its cluster, and if there isn't one, the algorithm should recognize that it harshly reduces fitness. Again, a median set size of 10 is used, and no known optimal exists. An image is shown below.



Large Dataset 3 (clustered)

Results

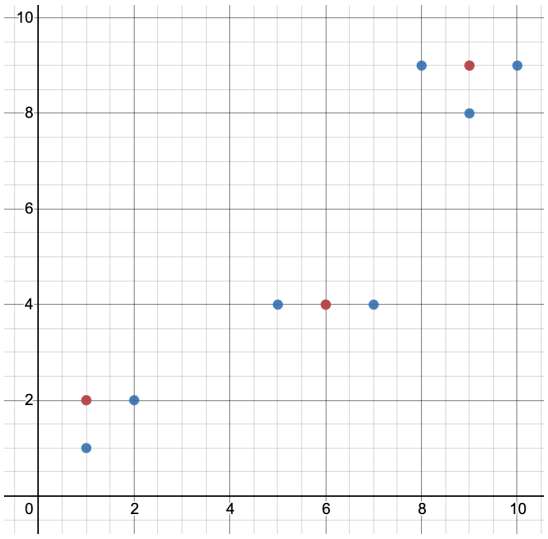
Results for each of the datasets are shown below. A table exists for each dataset that shows performance for each combination of operators for each algorithm. The table includes the best overall solution fitness for each combination, as well as average generation number and average best solution. SA/HC data does not have generation numbers, as they always run for 71,255 iterations. Each combination was run 5 times for each dataset so that an average across 5 trials could be calculated.

Toy Dataset Results

GA						
	Selection	**Crossover**	**Mutation**	**Best Overall**	**Avg. Generations**	**Avg. Best**
	Roulette	Double Point	Single Point	7	102.4	7
	Roulette	Double Point	Nearest Neighbor	7	100	7
	Roulette	Uniform	Single Point	7	100	7
	Roulette	Uniform	Nearest Neighbor	7	100.8	7
	Tournament	Double Point	Single Point	7	102.2	7
	Tournament	Double Point	Nearest Neighbor	7	102.2	7
	Tournament	Uniform	Single Point	7	100.6	7
	Tournament	Uniform	Nearest Neighbor	7	102.2	7

SA			
	Perturbation	**Best Overall**	**Avg. Best**
	Single Point	7.000	8.380
	Nearest Neighbor	7.000	24.175
Foolish			
	Perturbation	**Best Overall**	**Avg. Best**
	Single Point	7	7
	Nearest Neighbor	7.000	9.298

For the toy contrived dataset, each combination was able to find the solution, which had a fitness of 7. For the GA, each combination found the solution in all 5 trials, within very few generations (Note: the termination condition forces at least 100 generations, so these took on average less than 3 generations to find the optimal). For SA/HC, each combination found the optimal in at least one trial. Foolish with Single Point found it in all five, but the others had some poor solutions. For SA with Single Point, this is likely due to taking a bad move late enough in the process that it never gets fixed. For SA/HC with Nearest Neighbor, there appears to be more trouble. This will get discussed later, as the trend continues with the other datasets. The best solution found (the optimal) is shown graphically below, with red points being the median set.



Toy Dataset Solution

Medium Dataset Results

GA						
	Selection	**Crossover**	**Mutation**	**Best Overall**	**Avg. Generations**	**Avg. Best**
	Roulette	Double Point	Single Point	674.492	215.400	675.563
	Roulette	Double Point	Nearest Neighbor	674.492	144.600	677.880
	Roulette	Uniform	Single Point	674.492	200.600	675.044
	Roulette	Uniform	Nearest Neighbor	674.492	164.200	678.112
	Tournament	Double Point	Single Point	692.526	110.000	722.479
	Tournament	Double Point	Nearest Neighbor	683.531	170.400	703.764

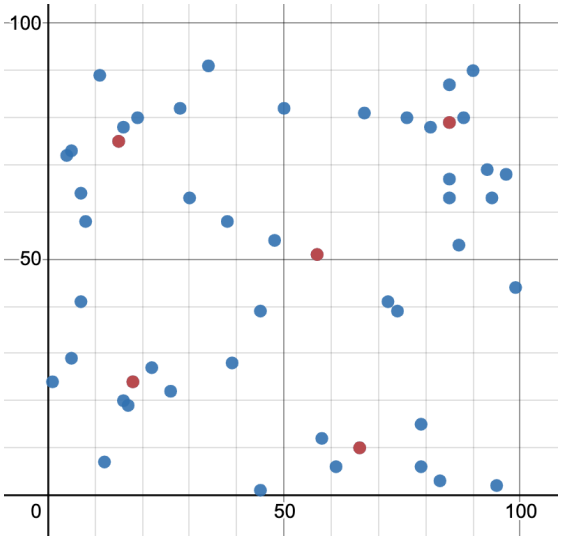
	Tournament	Uniform	Single Point	697.880	133.200	713.739
	Tournament	Uniform	Nearest Neighbor	695.488	137.000	712.509

SA			
	Perturbation	**Best Overall**	**Avg. Best**
	Single Point	674.492	675.663
	Nearest Neighbor	726.496	1166.890
Foolish			
	Perturbation	**Best Overall**	**Avg. Best**
	Single Point	674.492	674.492
	Nearest Neighbor	799.201	1255.888

For the medium dataset, no known optimal was known. The best overall solution fitness is highlighted in the table. Since many combinations found the same solution, it is possible that it is the optimal solution for this dataset. Notable data for average generations and average best for those that found this potential optimal are highlighted as well. The Roulette, Double Point, Nearest Neighbor combo found the competing ‘optimal’ in the fewest generations, while the Roulette, Uniform, Nearest Neighbor combo had the best average result. Foolish Hill-Climbing with Single Point notably found the same answer (tying with the other bests) in all 5 trials.

Notable trends in this data are that Tournament selection performed consistently worse than Roulette, though took fewer generations to converge. Also, within the Tournament combinations, Nearest Neighbor mutation provided a slight edge to Single Point, and Double Point had an edge on Uniform. For SA/HC, Nearest Neighbor performed significantly worse than Single Point, especially in the average case.

The best solution found for this dataset is shown below (found by many combinations). The median set consists of the points [(18, 24), (66, 10), (57, 51), (85, 79), (15, 75)].



Medium Dataset Best Solution

Large Dataset 1 Results

GA						
	Selection	**Crossover**	**Mutation**	**Best Overall**	**Avg. Generations**	**Avg. Best**
	Roulette	Double Point	Single Point	5424.624	358.000	5512.651
	Roulette	Double Point	Nearest Neighbor	5397.007	272.800	5545.748
	Roulette	Uniform	Single Point	5428.341	329.400	5518.891
	Roulette	Uniform	Nearest Neighbor	5393.224	345.600	5480.496
	Tournament	Double Point	Single Point	5770.543	206.800	5971.541
	Tournament	Double Point	Nearest Neighbor	5888.948	142.800	6077.260
	Tournament	Uniform	Single Point	5880.633	113.400	6094.593
	Tournament	Uniform	Nearest Neighbor	6060.593	121.600	6140.022

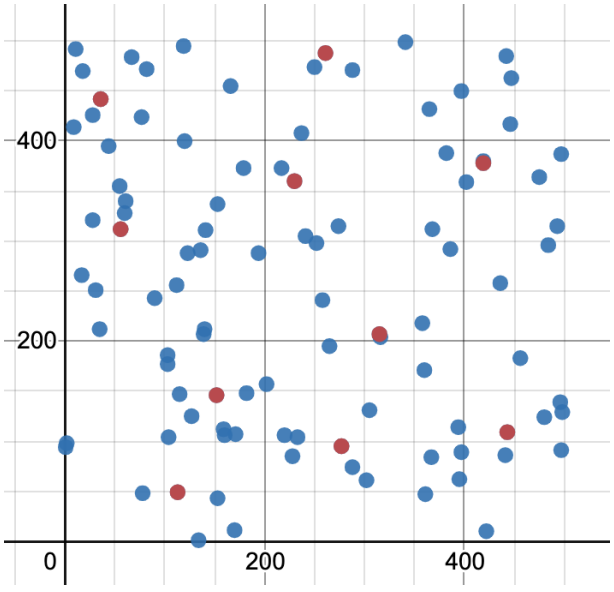
SA			
	Perturbation	**Best Overall**	**Avg. Best**
	Single Point	5334.751	5343.239
	Nearest Neighbor	5862.137	7052.107
Foolish			

	Perturbation	Best Overall	Avg. Best
	Single Point	5334.697	5349.887
	Nearest Neighbor	6930.291	7848.591

For Large Dataset 1, the best solution found by the GA was with Roulette, Uniform, Nearest Neighbor, which also performed best on average. However, the best solution overall was found by Foolish with Single Point. SA Single Point's best was just behind Foolish Single Point, although it did have the best average performance.

For this dataset, SA/HC with Single Point outperformed the GA, though GA with Roulette was closed behind across the board. Roulette continued to outperform Tournament, though Tournament had faster convergence. Interestingly, Nearest Neighbor seemed to improve Roulette results, but not for Tournament. Additionally, Double Point seemed better than Uniform with Tournament selection, but the two crossovers seemed similar for Roulette. Again, Nearest Neighbor proved to be poor as a perturbation for SA/HC.

The best solution found for this dataset, which was found by Foolish with Single Point, had a median set of [(152, 146), (113, 49), (230, 360), (56, 312), (36, 442), (315, 207), (443, 109), (419, 378), (277, 95), (261, 488)], and is shown below.



Large Dataset 1 Best Solution

Large Dataset 2 Results

GA						

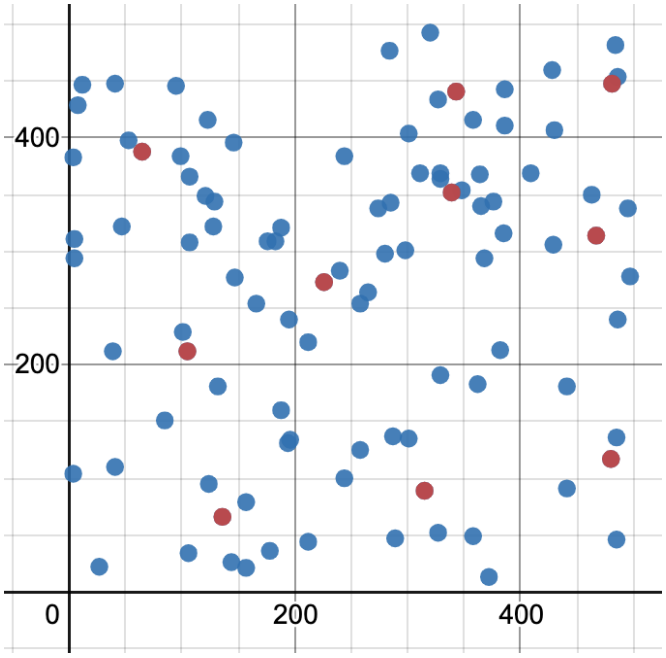
	Selection	Crossover	Mutation	Best Overall	Avg. Generations	Avg. Best
	Roulette	Double Point	Single Point	5321.598	340.800	5417.839
	Roulette	Double Point	Nearest Neighbor	5369.700	235.400	5496.803
	Roulette	Uniform	Single Point	5410.547	344.000	5487.109
	Roulette	Uniform	Nearest Neighbor	5428.969	309.200	5467.766
	Tournament	Double Point	Single Point	5600.458	184.200	5933.080
	Tournament	Double Point	Nearest Neighbor	5577.621	244.000	5700.330
	Tournament	Uniform	Single Point	5918.167	123.800	6011.465
	Tournament	Uniform	Nearest Neighbor	5865.339	113.800	5973.063

SA			
	Perturbation	**Best Overall**	**Avg. Best**
	Single Point	5322.248	5343.063
	Nearest Neighbor	6263.361	6589.445
Foolish			
	Perturbation	**Best Overall**	**Avg. Best**
	Single Point	5322.248	5349.270
	Nearest Neighbor	6561.510	7196.820

The best solution found for Large Dataset 2 was found by the GA combination Roulette, Double Point, Single Point, which also had the best average for GA, while SA Single Point had the best average of all algorithms. SA and HC with Single point both found the same best solution in their trials, though SA did slightly better on average.

This dataset continues the trend between Roulette and Tournament, as well as the failures of Nearest Neighbor with SA/HC. In this data, Double Point crossover performed better than the corresponding Uniform crossover combos. Nearest Neighbor seemed to slightly improve Tournament combos, but showed a small decrease for Roulette combos.

The best solution found for this dataset (found by Roulette, Double Point, Single Point) had the median set [(226, 273), (315, 89), (136, 66), (343, 441), (105, 212), (65, 388), (481, 448), (467, 314), (339, 352), (480, 117)], and is shown below.



Large Dataset 2 Best Solution

Large Dataset 3 Results

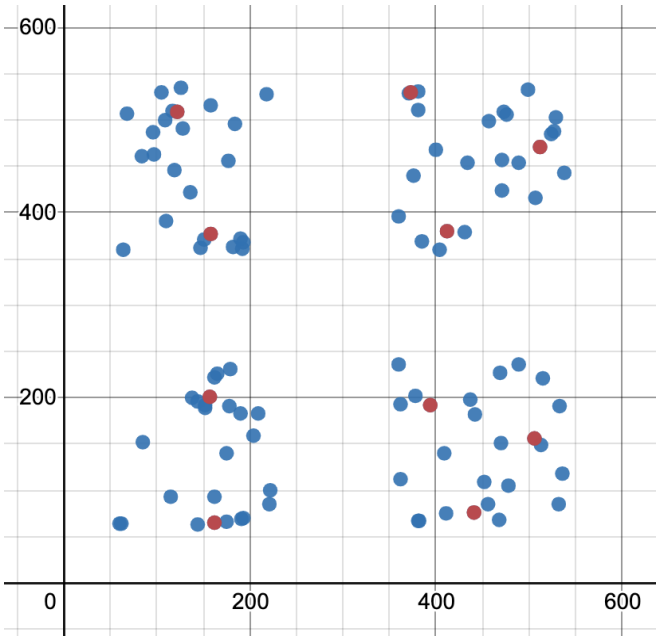
GA						
	Selection	**Crossover**	**Mutation**	**Best Overall**	**Avg. Generations**	**Avg. Best**
	Roulette	Double Point	Single Point	3938.067	474.200	3956.346
	Roulette	Double Point	Nearest Neighbor	4049.877	275.200	4093.302
	Roulette	Uniform	Single Point	3934.545	426.200	3980.801
	Roulette	Uniform	Nearest Neighbor	3934.545	357.800	4004.441
	Tournament	Double Point	Single Point	4299.761	145.600	4516.269
	Tournament	Double Point	Nearest Neighbor	4623.904	150.600	4756.028
	Tournament	Uniform	Single Point	4300.429	165.200	4601.432
	Tournament	Uniform	Nearest Neighbor	4608.857	111.400	4756.030

SA			
	Perturbation	**Best Overall**	**Avg. Best**
	Single Point	3932.356	3934.209
	Nearest Neighbor	6251.397	7781.570
Foolish			
	Perturbation	**Best Overall**	**Avg. Best**
	Single Point	3932.356	3932.356
	Nearest Neighbor	4688.048	6815.205

Large Dataset 3 is the clustered dataset, and the algorithms seemed to do well at dispersing medians amongst the clusters. The best solution found for GAs was found by both Roulette, Uniform, Single Point and Roulette, Uniform, Nearest Neighbor, with the former doing better on average but the latter in fewer generations. Notably, Roulette, Double Point, Single Point had the best average, but did not find as good of an overall best. For SA/HC, both with Single Point found the same best solution, which is also the best of all the algorithms, and Foolish HC did so all 5 times. It is possible HC got stuck at a local optimum, but since no other combination could beat it, it may have found the optimal solution.

The Roulette/Tournament trend continued here, as well as Nearest Neighbor's SA/HC failure. Double Point performed comparably to Uniform. Nearest Neighbor performed noticeably worse than Single Point when used with Tournament, but comparably when used with Roulette.

The best solution found for this dataset was found by both SA and Foolish HC with Single Point, and had a median set of [(162, 65), (157, 201), (394, 192), (441, 76), (506, 156), (373, 530), (512, 471), (412, 380), (122, 509), (158, 377)], and is shown below.



Large Dataset 3 Best Solution

Conclusions

In conclusion, each of these three (GA, SA, HC) seem to have potential in effectively solving this problem. They were all competitive with each other, and even when one performed better, the others were usually close behind.

For the selection operators, Roulette consistently outperformed Tournament. In all cases it provided better solutions, though it took more generations to converge. It appears that Roulette does better to propagate good genes that eventually lead to strong solutions.

For the crossovers, both Double Point and Uniform proved effective. Double Point seemed to do slightly better most of the time, but not always. In any case, the difference was fairly minimal, suggesting that both of these crossovers function adequately.

For mutation, Single Point and Nearest Neighbor both performed well. Which one performed better seemed to be dependent on the dataset and which selection operator was used. Both appear to have their merits for being used as a mutation operator in a GA.

For perturbation in SA/HC, Single Point dominated over Nearest Neighbor. Single point offered strong, competitive solutions in all datasets, while Nearest Neighbor

struggled (even in the toy dataset). This makes sense, as these algorithms can be heavily dependent on the initial solution, and Single Point does a much better job exploring the search space. Since Nearest Neighbor makes a very minimal change, and because it has the potential to get stuck switching medians only between a few close sets of points, it easily gets stuck in the search space and cannot explore effectively. The data suggests that Nearest Neighbor may be a good mutation operator in GAs, but when it is used as the sole perturbation for exploration it fails.

One detail that I found while implementing these solutions and seeing their results was that Simulated Annealing would often find a good solution in the middle of its iterations, but then move away from it and never recover to a more optimal solution. Because of this, I would think that one simple way to improve the algorithm is to keep track of both the best solution it has found so far and still letting it replace its current solution state with a worse one for more exploration. This way if it finds a good solution early, it can continue to explore without losing that potential answer.

Overall, this project was interesting and enjoyable. I really enjoy seeing how mimicking nature can lead to effective algorithms for solving hard problems.