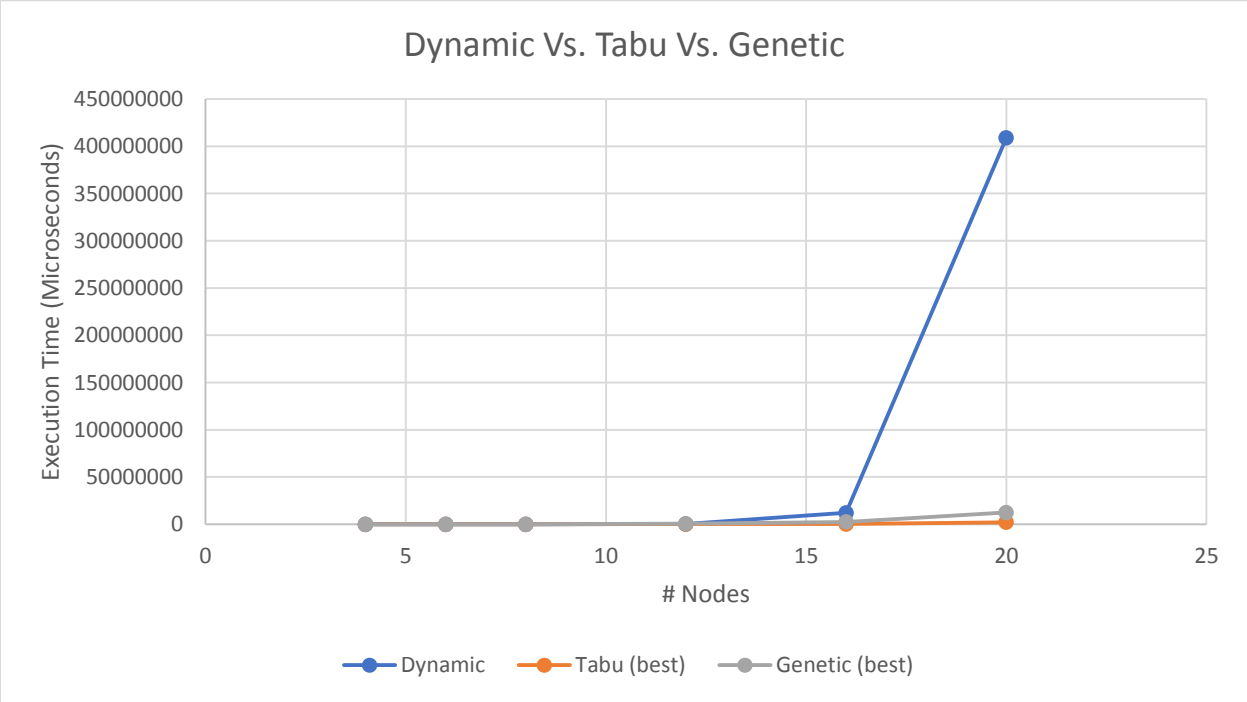
Hayden Donofrio

CSE 3353

LAB 04 Report

Overall Comparison:

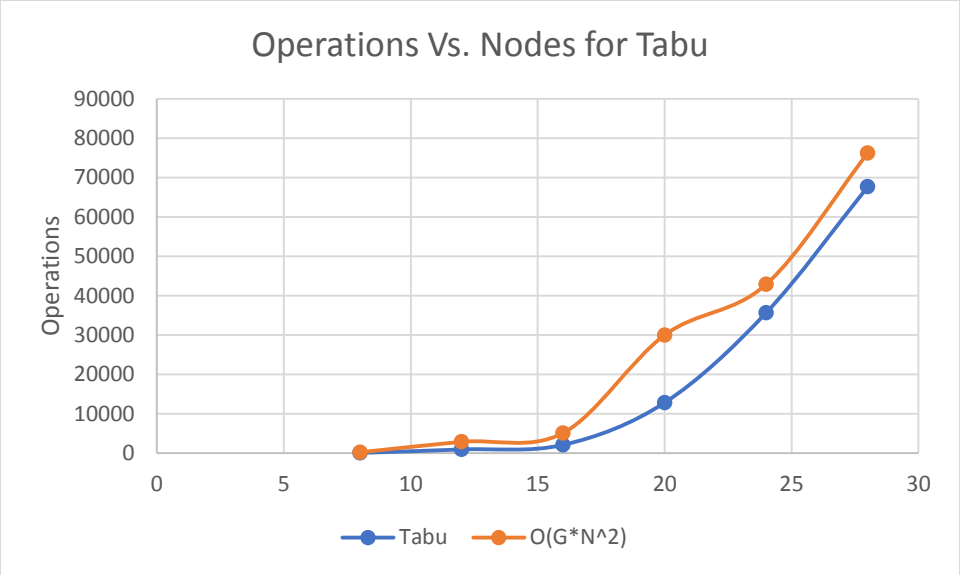
In this section I will be comparing the overall performance of tabu vs. genetic Aagorithm vs. dynamic programming for TSP. For TSP a basic knowledge of setting up the graphs and knowing the search space will have a huge impact for how quickly an algorithm will perform on a specified data set. We will see that in tabu especially since I began tabu with a “good” starting solution. (moving to the closest city from each point that hasn’t been visited.) genetic however, starts with a completely randomized population and works its way up from that.

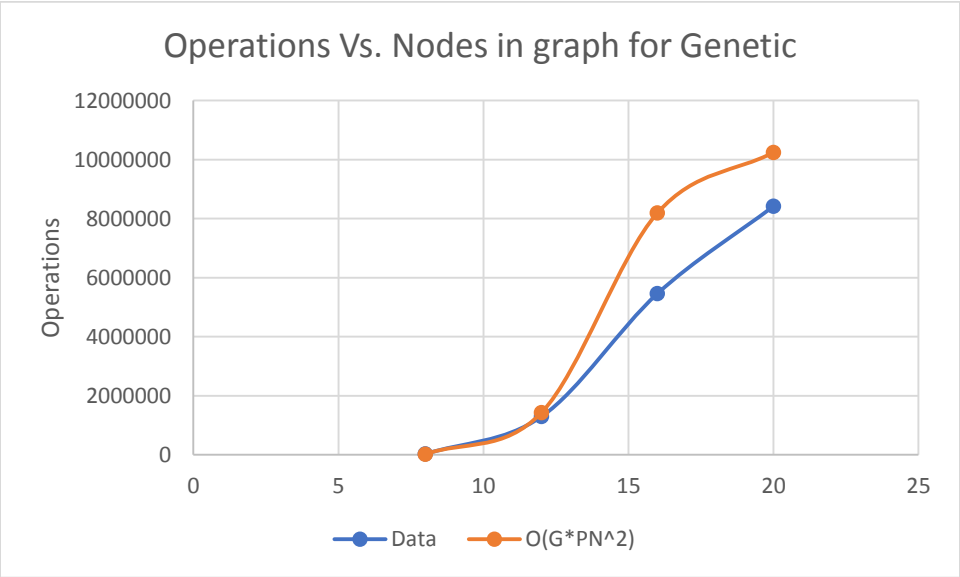


As we can see here, to reach an optimal solution it takes the dynamic programming approach significantly more time than both the tabu and genetic search. Something that I found

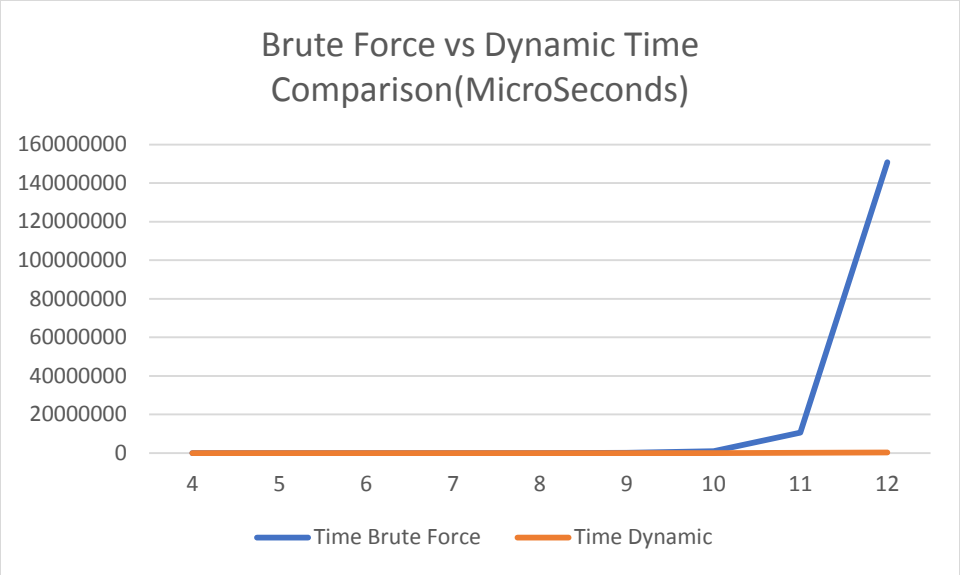
really surprising was that genetic was outperformed by the tabu search at the operation termination. I conclude that this is because of the starting point with tabu search: we already know a good solution to start with based on the knowledge of the search space, which makes the Ttbu search find an optimal or better solution very quickly, therefore the tabu algorithm stopped sooner because it reached the optimal solution that I found using dynamic programming.

Genetic algorithm also has many more operations to do than tabu does. The only tedious operation in a traditional tabu implementation is searching the tabu list to see if the current solution exists. This can be solved by keeping the list size at an appropriate limit: one that is not too big as to include too many operations, and one that is not so small that graphs stay stuck on plateaus or other areas such as local minimums, and the search space keeps moving. Genetic has to calculate probabilities, select out of those probabilities, and create children. This can be improved by using a population size that is not too big where the operations become tedious and not so small such that we do not see a very different population change between generations. I have changed many different kinds of population sizes in genetic, and the amount of operations vs. the amount of time needed is more determined on a case-by-case bases. If one wanted a very good solution, we should increase the population.

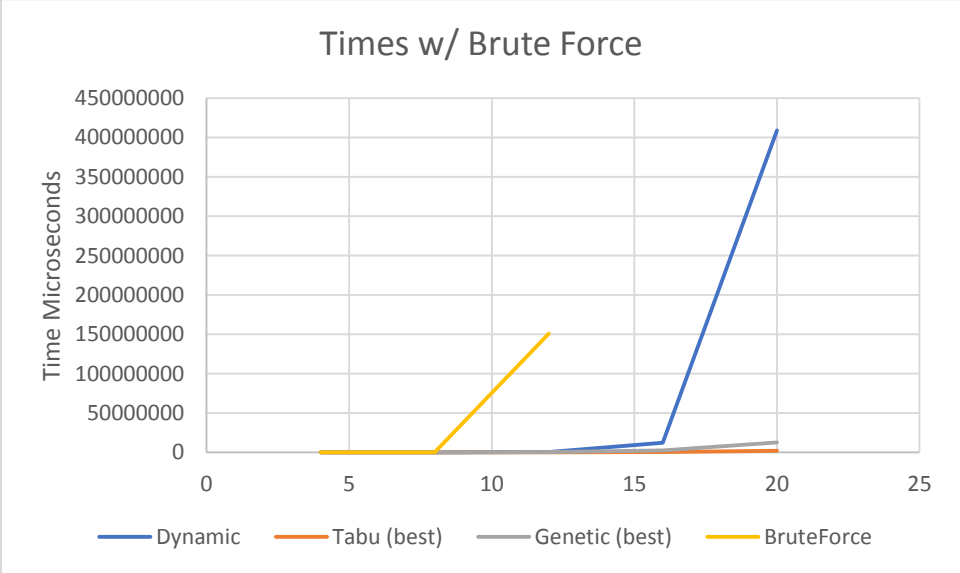




Here are the complexities of the two graphs. The difference here is the amount of generations required for a good convergence. Due to the way my tabu neighborhood is written, it takes fewer generations as we get larger to find a good solution. However, as we will discuss, genetic is much better for larger sets of data due to its non-convergent nature.



Here is an old graph comparing Dynamic to Brute Force. Genetic and Tabu would not even appear on this graph since as we can see brute force performes much worse than dynamic at a factorial scale. Compared to the first graph, this shows how signifcant these new algorithms are for acheiveing a good solution in a signifcantly smaller amount of time.



Including brute force into the original graph, we can see that the slop between two points increases at a much higher rate than all of the others. I was not even able to put a point

for 14 or 16 because it would have taken hours to run. Therefore, we can see that tabu and genetic extremly outperform the brute force approach with time.

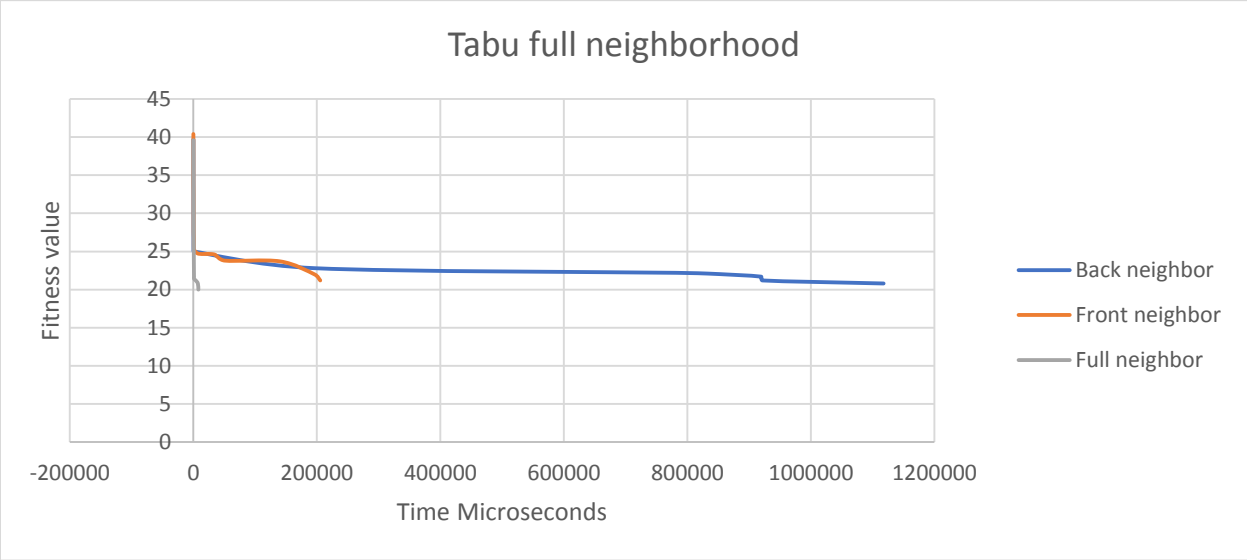
For the graphs of genetic and Tabu I used the combination of neighborhoods, selection, mutations, etc.. that most often provided the best answer. Since there is no defined stopping point for the meta-heuristic searches it is very difficult to tell what their complexity analysis is. However through the amount of operations performed on the average data set of 20 nodes gives us a taub complexity of O(G*N^2) to find an improved solution for tabu search. This is related to G generations and each generation performs about N squared operations to return the best neighbor. and genetic is O(G*PN^2) where N is the size of the graph and G is the number of generations and P is the number of chromosomes in the population. My version of genetic algorithim has many more operations to do, so unfortunately it performs slower sometimes realtive to which selection or comparison we are doing. However, as we will find, some versions of genetic will continue to search the total search space while tabu has a difficult time avoiding convergence.

I have three versions of neighborhood selection for Tabu.

1. Full neighborhood swap: any two nodes can swap with each other
2. Front neighborhood swap: the first non-fixed node can swap with any other node
3. Back swap: the last non-fixed node can swap with any other node.

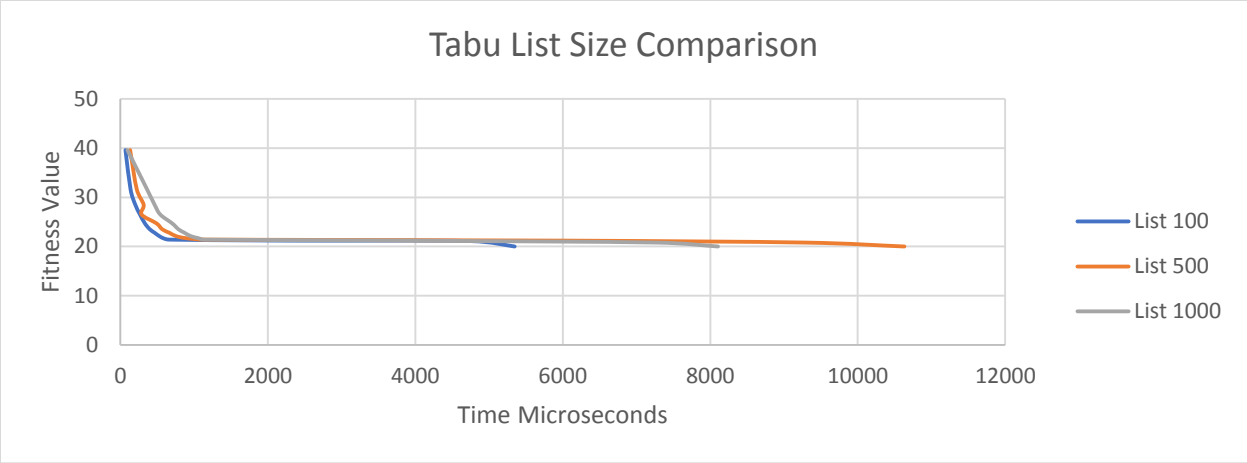
I predicted a lot more calculations for the full neighborhood swap, but it would find a good solution in fewer loops. Therefore, I thought that it would perform about the same as the other two options. However, the full neighborhood swap severly outperformed the other two neighborhood types by a signifcant factor.

(ALL TESTS WITH A GRAPH OF SIZE 20, OPTIMAL SOLUTION 21)



It may be difficult to see, but see how long it takes full neighbor to come to the optimal solution compared to the other two, front neighbor of which converges to an incorrect solution, and back neighbor of which terminates because of operations. It looks like the front neighbor converged to a certain solution (or large plateau) and could not find its way out while the back neighbor eventually found a way out of convergence. Because of this I have concluded that the full neighbor swap is by far the most efficient. The other two behave at a timing similar to genetic algorithim, which we will discuss later.

For this particular search space, I have found that smaller tabu lists generally outperform larger tabu lists, especially for the full neighbor swap.



All of these eventually reach the optimal solution. However, the 100 sized list performs much better than the other lists. This may just be because the amount of operations to search through the tabu list takes signifcantly less time since it is a linear search. Therefore, Tabu search generally perfers smaller lists for short term memory. In the future we may want to incorperate another list for long term memory to stop overall converging at large plateaus.

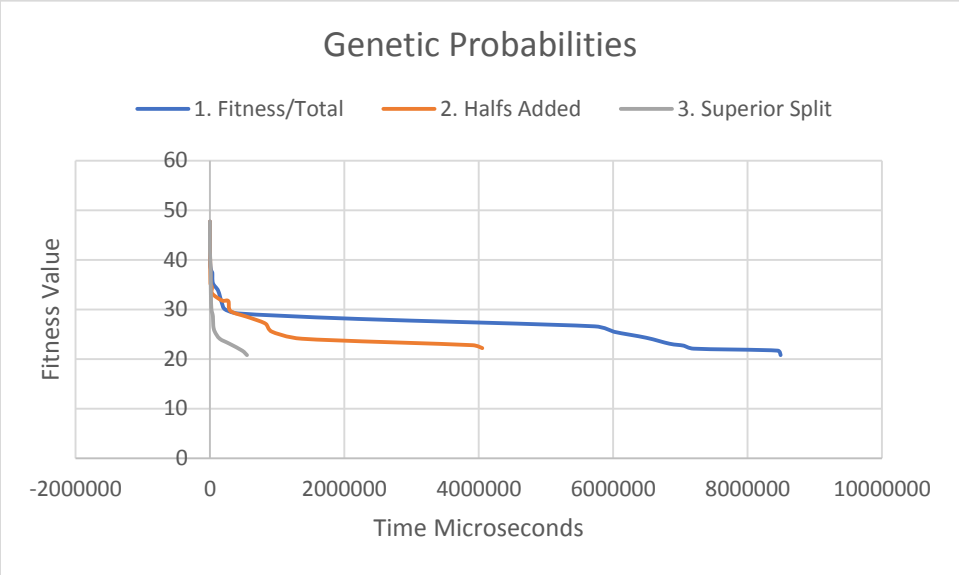
GENETIC

My version of genetic algorithim is fairly resource heavy. However, it usually finds a very close or optimal solution eventually. I think this is because of the amount of iterations I go through for each generation. If time was of no serious concern (a couple of minutes for a large graph, 50+) and one wanted to ensure that eventually it would converge to a good solution, I would heavily suggest genetic alogirhtim over tabu. However, for our purposes, we will only be comapring our genetic algorithm to itself in this section over a graph of 20.

For probabilites I had three different ways I calculated it. I used a general roulette wheel selection for all three types, these are just the following ways I selected probabilities for roulette wheel.

1. Fitness Value/ overall fitness. This was the least differentiated.

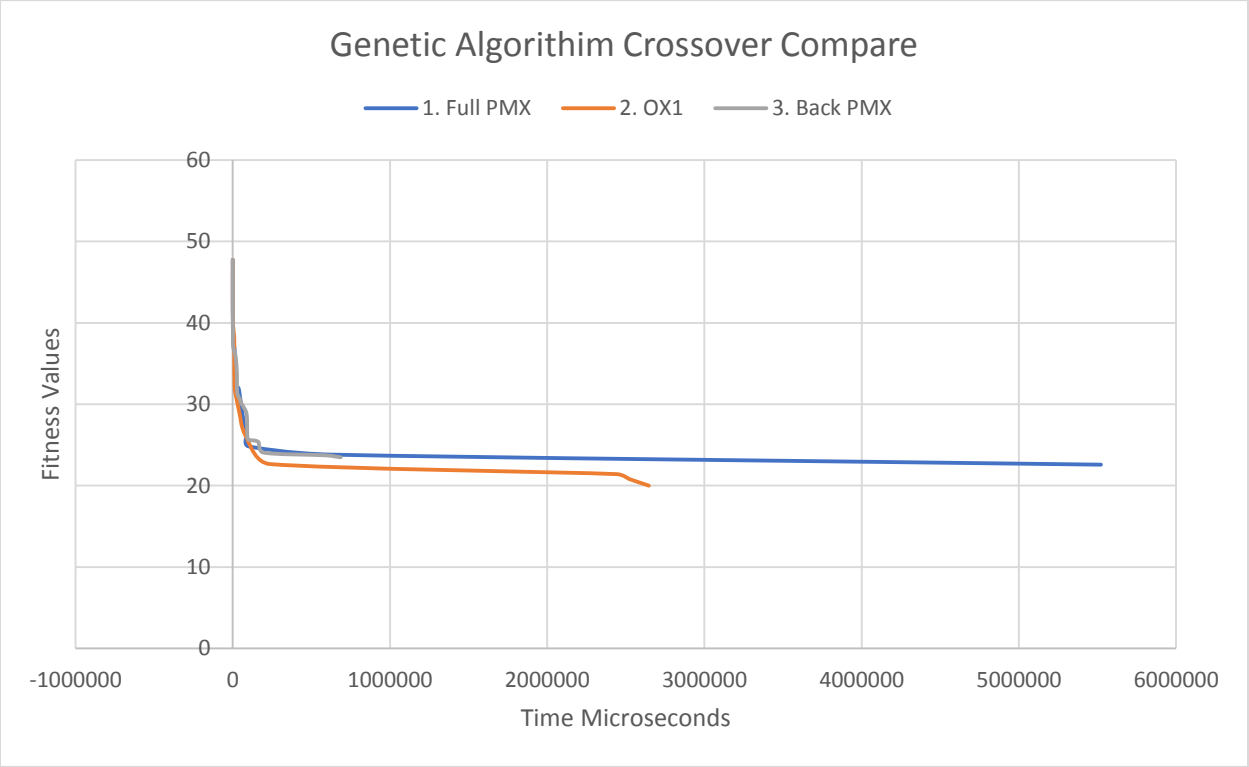
2. Fitness Value/ overall fitness + the highest percentages also took half of the value from the lower half of percentages. I.e. for a population of 10, 1 (the highest percentage) also took 50% of 6, 2 took 50% of 7, etc…
3. Fitness Value/ overall fitness + strong leading from the lower fitness values. I.e. imagine another population of 10, 1 took 100% of 10, 2 took 75% of 9, 3 took 50% of 8, etc..



Method 3 takes the cake here. It signifcantly outperforms the other two methods, this is because our stronger parents have the highest chance of being selected, but there still is a good chance for selection of others as well.

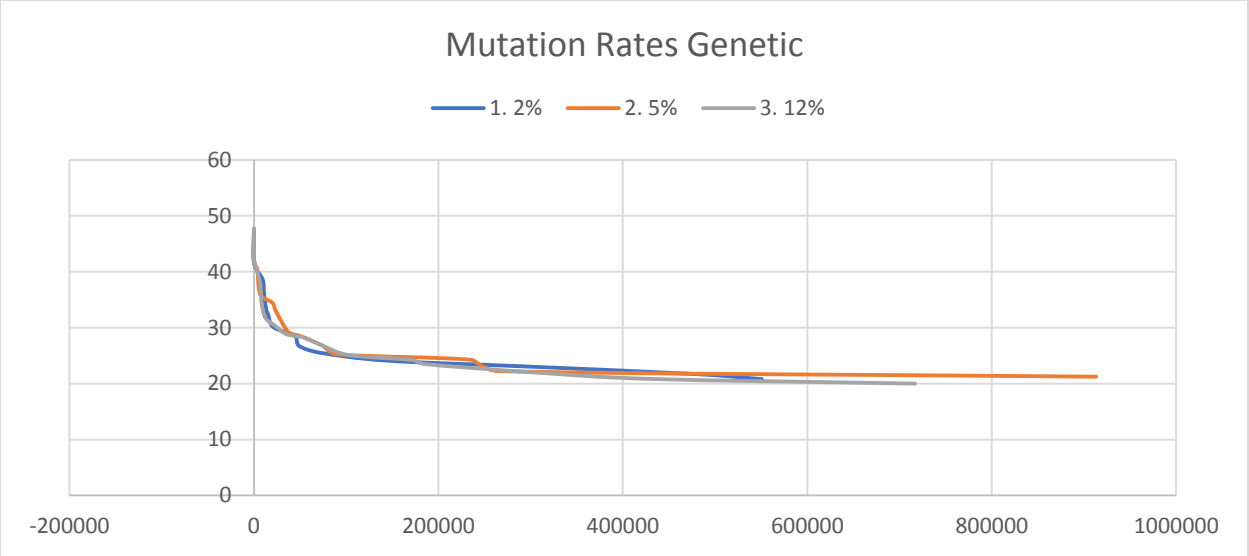
For selection types I had three different types.

1. Standard PMX
2. Order 1 Crossover
3. PMX, but only one element is swapped onto the stronger parent (basically selecting the strongest parent)



In the following graph it is shown that 1 order based crossover generally found a good solution faster than the others. It actually found the optimal solution and terminated. However, Back PMX basically struggled and converged to a certain point, which proves that always selecting the strongest parent is never a good idea for large search spaces. PMX never found the optimal solution but never converged, this would be an okay solution for large search spaces, but it is reccomended to stay with order based crossover.

For mutation rates I used 2, 5, and 12 percent. A mutation just randomly swapped two nodes in the chromosome.

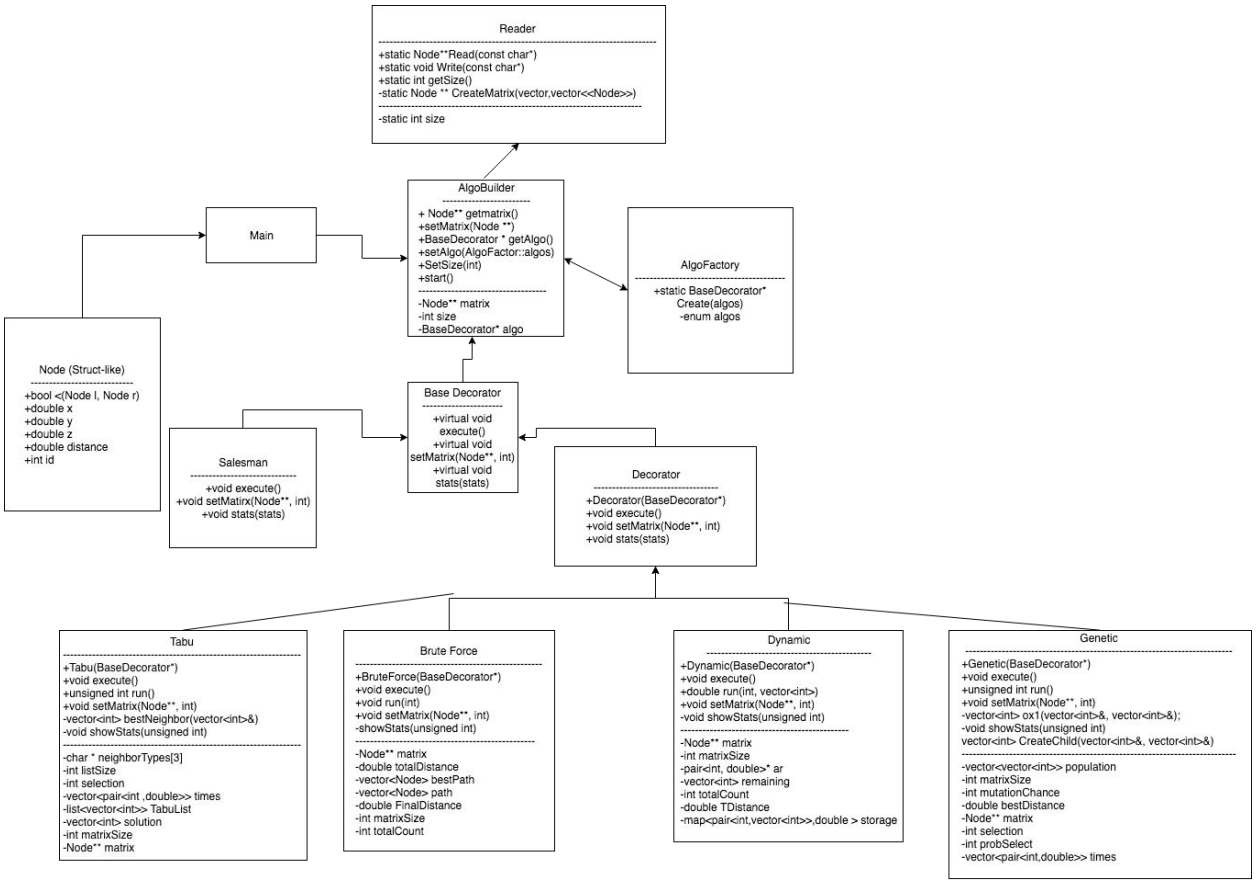


Most of the mutation rates show fairly similar results, with 12% just barely beating out the others, it is the only one that found the optimal solution before termination. This shows that a good mutation rate for the traveling salesman problem should not be too low, it should encourage random changes to help improve the population. The higher mutation rate generally reached a much better solution at each point in the graph and had a less of a chance of converging, while lower percentage rates stuggled and eventually converged.

Overall, I was very surprised how well tabu did, such a simple algorithim can perform very well with a little knowledge of the search space. As I continued to increase the data size I tested with, including results not shown here, it proved that genetic almost never converged with PMX or first order crossover, so If I was searching a much larger graph, I would sacrifice time for overall performance with genetic. If I was searching any smaller graphs, I would easily choose tabu due to how quickly it can find a good solution.

DESIGN:

I took almost the same exact design as before in lab 3. A factory that is used by a builder class which is essetially an interface for a decorator that uses my different algorithim types. It was very easy to implement some methods in the decorator and use them in the child classes which was a good sign since I did not have to re type my code. I will continue to use this pattern In the next lab due to how modular it is to send the graph around to different algorithims and how easy it is to design an algorithim and put it in the decorator class for use with my builder.



The inheritance from the decorator made it very easy to send some methods over such as calculating distance, which is vital to both of these methods because

this is how fitness is calculated. For particle swarm and simmulated annealing, I will use the same decorator and salesman interface to display stats and calculate fitness scores.