Lab 3 Results for Brute Force and Dynamic Programming timing.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Nodes | Brute Force Average Time | Dynamic Programming Average Time |
| 1 | 0 | 0 |
| 2 | 0.001 | 0.001 |
| 3 | 0.001 | 0.001 |
| 4 | 0.001 | 0.001 |
| 5 | 0.001 | 0.001 |
| 6 | 0.002 | 0.002 |
| 7 | 0.004 | 0.003 |
| 8 | 0.021 | 0.006 |
| 9 | 0.079 | 0.012 |
| 10 | 0.6 | 0.1 |
| 11 | 8.01 | 1.13 |
| 12 | 93.8704 | 13.3698 |
| 13 | 1197.89 | 181.434 |

Lab 4 Results for Genetic Algorithm and Tabu Search timing.

|  |  |  |
| --- | --- | --- |
| Nodes | Genetic Algorithm Average Time | Tabu Average Time |
| 20 | 0.003 | 0.001 |
| 200 | 0.007 | 0.009 |
| 500 | 0.036 | 0.096 |
| 1000 | 0.144 | 0.966 |
| 1500 | 0.179 | 3.725 |
| 2000 | 0.294 | 5.735 |
| 2500 | 0.768 | 20.83 |

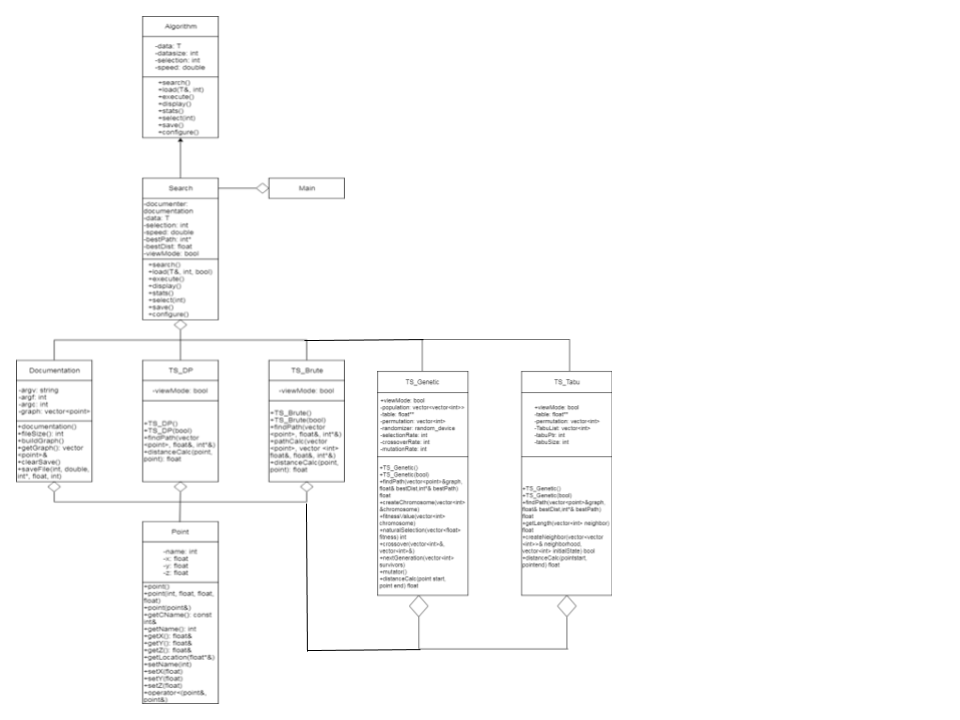
Genetic Algorithm (GA) is actually slower than Tabu (T) when the number of Nodes is low. But as the nodes increase, T becomes much slower than GA. This is due to the fact that GA requires a large amount of processing for the initial set up, but only requires a marginal amount of processing for each additional node added to the graph. By comparison, T does not require much processing for the initial setup but requires a greater amount of additional processing for each added node to the graph. GA manages to surpass T when the number of nodes approaches 200. These two algorithms were not compared to Brute Force or Dynamic Programming due to the fact that the difference between the four of them in terms of timing is so jarring that they are better compared separately. For instance, Brute Force and Dynamic Programming could not go above 12 nodes before the timing took too long to run to completion. In the case of GA and T, I needed to go to 20 nodes before either began to change in timing and took until 2500 nodes before T began to take too long to run.

One major difference between the two functions that was not able to be displayed in the graphs is accuracy. While Brute Force and Dynamic Programming did not need to be compared, since they both find the best-case scenario, GA and T do not necessarily guarantee to find the best solution. In fact, of the two, Tabu gets closer to finding the best solution than the Genetic Algorithm. My Genetic Algorithm tends to very quickly get stuck in local minimums and the crossover and mutation rates do not help it escape from these local minimums very easily. The Tabu algorithm finds a good solution with its initial greedy algorithm, and gets closer through the neighbor searches, but still does not quite manage to reach the best solution often. In general though, Tabu performs better than Genetic Algorithm, but takes a larger amount of time. Genetic Algorithm performs great, even with a high number of nodes, but the tradeoff for speed seems to be accuracy, at least in the case of the algorithms I created.

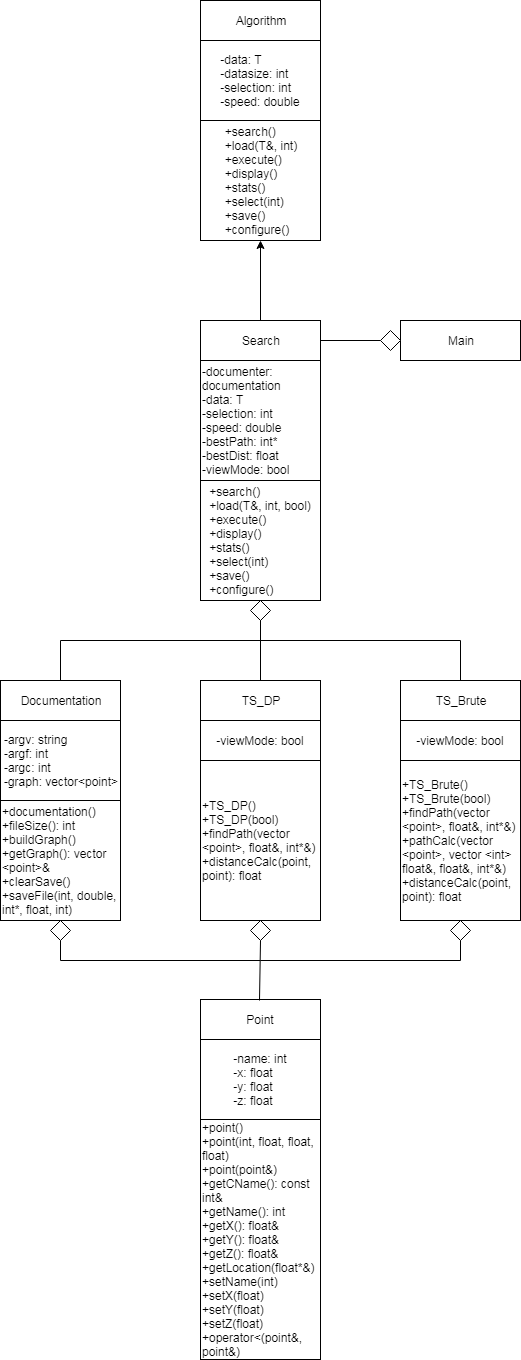
When adjusting The Selection Rate for GA, the higher the Rate, the more generations pass before it eventually converges into a local minimum. It does not seem to drastically improve the accuracy of the final solution, but it does have the largest impact out of the other techniques. Adjusting Mutation Rate requires a very large difference before its presence begins to make a difference in the solution. The biggest problem that mutations tend to have is that it alters so little of the actual path, especially as the number of nodes increases to really make a difference. When the number of nodes is below 15, Mutation Rate can actually help GA break out of local minimums and get a better solution. But if the number of nodes is above 15, Mutation Rate begins to have almost no impact at all, even at 100%. Crossover Rate performs a similar function to Selection Rate, in that the larger it is, the longer it takes before it converges. However, the higher this number is, the longer the timing takes before it finishes the program. This is likely due to that fact that Crossovers are called far more often than the Selections. In conclusion, Selection Rate provides the best overall performance increase, so long as it is below 5, but Mutation Rate works best for smaller graph sizes, and Crossover Rate trades a slight performance increase in exchange for a large timing increase.

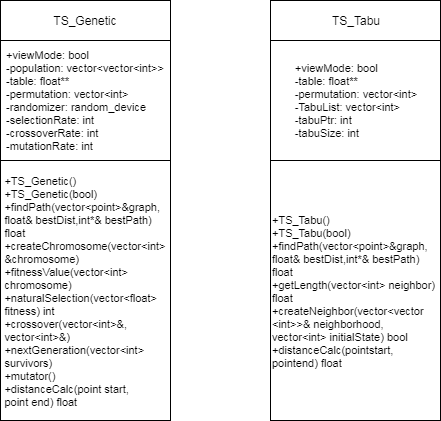
Tabu has a much more black and white approach. The larger the Tabu List is set at, the longer the program takes to complete, since the list is used to determine when to stop creating neighbors. Since Tabu gets close enough to the best solution anyways, increasing the Tabu List size above 20 increases the timing of the programming even higher, but does not seem to actually improve accuracy all that much. In a way, it is very similar to increasing Crossover Rate from the GA. It trades better accuracy for worse timing, except that Tabu is already slower and more accurate than GA, so I do not see a reason to increase the Tabu List size to more than 10-20.

System Architecture:



(Quality is very low in this image, so I split it into two below for better clarification)

(Lab 3 Architecture)

(Lab 4 New Classes)

Like the previous lab, I only wanted to include one class into the main class. Due to how I set up the previous lab, I was able to keep this goal and only needed to add two new classes to the program to get it to work properly. The two new classes are TS\_Genetic and TS\_Tabu. Both of these include the Point class and are included by the Search class. No new functions were needed to be added to any of the previous classes. Only 30 lines of new code was added to the Search class and 6 lines were added to the Documentation class. Within the main class, only 8 lines of code was added. In total, only 42 lines of code were added to other classes outside of two new classes added in order for the program to function.

On designs for the future, I would anticipate that it would only take 21 lines of new code for each algorithm added in the future, provided that it was an algorithm added to deal with the Traveling Salesman Problem. This is on top of the code needed to create the new class in the first place.