Colin Weil

Lab 4 Heuristic Search Report

**Algorithms**

SA - Simulated Annealing

GA - Genetic Algorithm

**Configuration**

**Selection:**

SR – Roulette

SRA – Random

**Mutation:**

SR – Random

SRA - Adjacent

**Cross Over:**

SR – Slice

SRA – Alternating

T – Starting Temperature

CC – Cooling Constant

**Nodes vs Time of Each Algorithm**

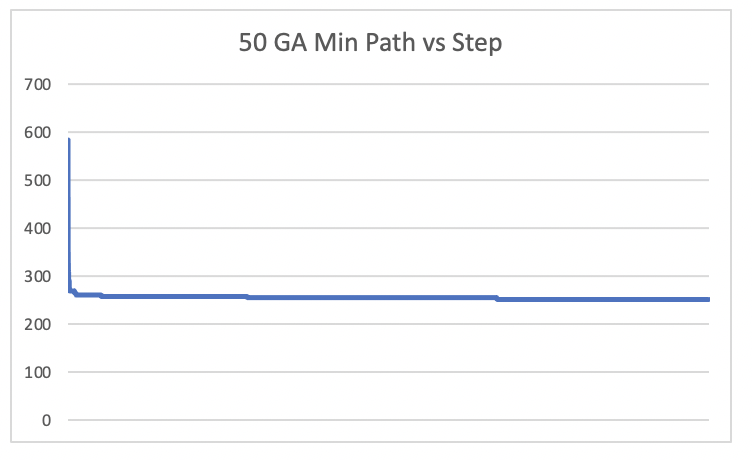
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nodes | Brute | Dynamic | SA (T = 10000, CC = .000001) | GA (SR\_MA\_CS) | O(n!) | 0.000005 O(n^2\*2^n) |
| 3 | 0.00002 | 1.2E-05 | 4.9457 | 0.175417 | 6 | 0.0000015 |
| 4 | 0.000057 | 0.00002 | 5.00659 | 0.26246 | 24 | 0.000002 |
| 5 | 0.000192 | 3.6E-05 | 5.2923 | 0.34267 | 120 | 0.0000025 |
| 6 | 0.000918 | 4.7E-05 | 5.35782 | 0.431769 | 720 | 0.000003 |
| 7 | 0.007179 | 0.00011 | 5.4478 | 0.569712 | 5040 | 0.0000035 |
| 8 | 0.046366 | 0.00017 | 5.72684 | 0.66391 | 40320 | 0.000004 |
| 9 | 0.326806 | 0.00035 | 5.9736 | 0.816452 | 362880 | 0.0000045 |
| 10 | 2.627307 | 0.00054 | 5.90529 | 0.933878 | 4E+06 | 0.000005 |
| 11 | 24.92624 | 0.0012 | 6.10301 | 1.1098 | 4E+07 | 0.0000055 |
| 12 | 274.720095 | 0.00258 | 6.11573 | 1.37609 | 5E+08 | 0.000006 |
| 13 | 3526.4484 | 0.00655 | 6.42137 | 1.46168 | 6E+09 | 0.0000065 |
| 14 |  | 0.01631 | 6.55257 | 1.58343 |  | 0.000007 |
| 15 |  | 0.04269 | 6.79903 | 1.59741 |  | 0.0000075 |
| 16 |  | 0.09652 | 7.95042 | 1.9142 |  | 0.000008 |

Lab 3 Algorithms:

Genetic Algorithm Configurations:

Simulated Annealing Configurations:

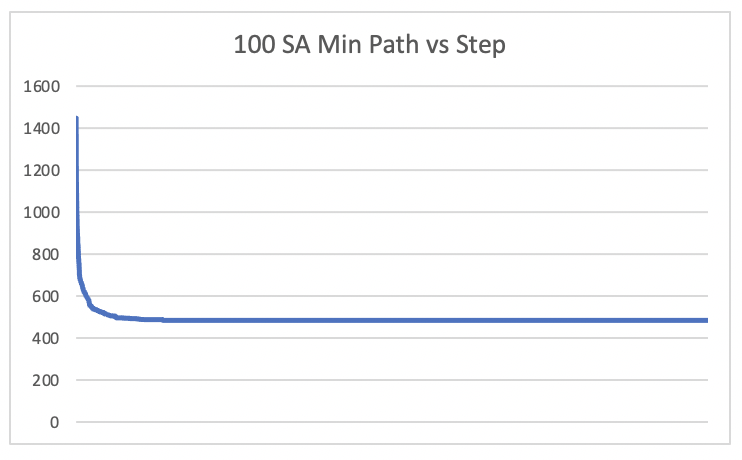
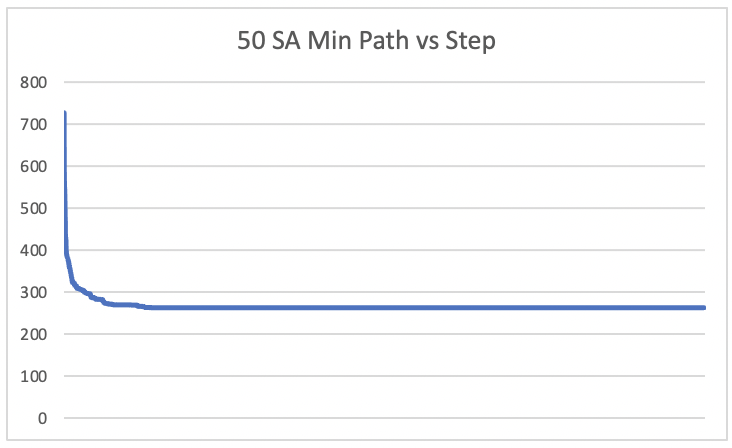
Genetic Algorithm 10 Minute Graphs (Data Size in Title):

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Simulated Annealing 10 Minute Graphs (Data Size in Title):

A picture containing chart

Description automatically generated

**Learning Curve Results:**

The general shape of the GA data seems to have more bumps than the smoother SA. This is most likely due to the simulated Annealing changing one random node with the other at a time, making small changes at a time. Genetic algorithm is quite the opposite. At the beginning, the crossover and selection can cause drastic change very quickly because there are so many components driving it to find the best path very quickly. Also, for both algorithms, it is an exponential function that nears a horizontal asymptote at the least expensive path. The starting random path, because of its randomness, is most likely not close to being the best. That is why it is very easy for these algorithms to find a better path quickly. Once the path becomes better and better, there are proportionally less good path to the total amount of paths, which will make the algorithms find the paths slower. Also, if you see in the 500 GA graph compared to the 500 SA graph, the SA graph has found a much better path. This is due to SA being more based on randomness, while GA only has a small chance of randomness helping the path. GA most likely was caught in a local minimum, and the mutations that were occurring weren’t pushing the best path in a better direction, so it stayed trapped in that local min.

**Variations in Genetic Algorithms and Simulated Annealing:**

Genetic Algorithm

Selection: The options for selection are roulette and random selection. The roulette is a hybrid between elitism and roulette because it will pick a random place in the population and then it loops through the population from that index until it finds a path that has a high enough fitness to be chosen. This has more diversity than elitism while still incorporating having two good paths could make a better path. The roulette version is obviously better for the selectin part than the random selection because random selection would just cause a brainless algorithm that didn’t incorporate the fitness or selecting anything better than before.

Mutation: The two mutations are adjacent and random. The adjacent will swap cities next to each other in a path and the random will switch random indexes in the path. The adjacent works better because if a path is already good, most likely that the general order is pretty close to being correct. This means that the adjacent mutation has a chance to make a small enhancement in the path while the random index has a higher chance to just not do anything beneficial.

Cross Over: The two cross-over methods are the slicing and alternating. Slicing will take a section from one parent then fill in the remaining nodes in order from the other parent. The alternating method will pick every other node from each parent in the order it shows up in the parent (if they are already in the path then it will skip that node). The slicing works better because by taking a big chunk that is from a selected parent means that chunk will probably be good. Then filling in with the other parent has a high chance of the other chunk being good, while also causing some variation and change in the path.

*It seems for GA, slight randomness is beneficial, but complete randomness is harmful for all methods of configuration.*

Simulated Annealing

Temp: The starting temperature is the y-axis on the learning curve graph. The temperature is a linear constant, so you need a big value to affect it a lot. The main purpose of it is that it controls the initial big decrease in path lengths, and then after the initial big decrease, which can be seen in the learning curve. As the temperature decreases, the chance of finding a better solution becomes smaller and smaller which means that the current best solution is close to being the actual best solution.

Cooling Constant: The cooling constant is the exponential constant that will decrease the temperature. I used a basic exponential equation to update my temperature every iteration, tempature = temperature\*(1-cooling constant). Although a simple equation, it was very effective in the cooling.

I found that using a very high temperature and then a very small cooling constant was very effective in finding the best path. Although this sacrificed running time, the combination of both almost always found the best path, even at path lengths.

**Design: (same thing as in lab report 3 except bolded part)**

Algorithm – I used a strategy pattern for this project. This allows me to pick an algorithm at run time and only create one object pointer in main. This means if I had other children of algorithm in my code folder, then I could also use them in main while easily switching between the algorithms but keep the naming conventions consistent because of the polymorphism between algorithm and its children.

HamiltonianCircuit – This class is a child of algorithm so it overrides all of algorithms methods. This class also has a function pointer in it with a vector of functions pointers that come from the static class, HamiltonianAlgorithms. When I turned in this code, there were only two methods/ solutions that I had made to this problem that were in the function vector and could be chosen by the Select function in HamiltonianCircuit, but I were to create another solution, then I could easily push it to the vector and choose select on it in order to use it in this class. **(I added the GA and SA algorithms by using this method which made it very easy to add one to what I already had)**

Path – Path is a simple class that contains a vector of ints that represent id’s in a created path as well as double of the length of the path. This class is used by HamiltonianCircuit and HamiltonianAlgorithsm in order to find and display the solutions.

fileReader – fileReader is the input class that can be added onto. For now, there is a 2D vector that holds the data of whatever type of file that needs to be read. The only type of file that needed to be read in for this project was the positions file. There is a function that receives a string of the file path of the positions.txt file, and then the file reader will parse the positions file and fill the 2D vector with the distance from every node to the other. More functions can be added for different file types in the future to populate the 2D vector.

output – The output class chooses a file path to send outputs to and then opens the file. The print function has a string parameter then prints the string to the output file. Since there were paths in this program, I also created a nicely formatted output for the paths (ex. 1->2->3). The output file can be expanded on for printing different types of items to a file in the future. **The output class also has a static function with ofstream by reference and a string parameter. This will print whatever string out to the output, and it is static so there doesn’t need to be an instance of the class to call it.**

**Hamiltonian Algorithms:**

At the top of HamiltonianAlgorithms.cpp, there are many options to be chosen from. There are constants that describe what the choices are and then there are also the constants that are used in the code that should be switched when wanting to use a different configuration of an algorithm. For GA, there are 2 options for: selection, mutation and crossover. For SA, the starting temperature can be changed by changing TEMP and the cooling constant can be changed.

**UML Diagram**:Diagram

Description automatically generated