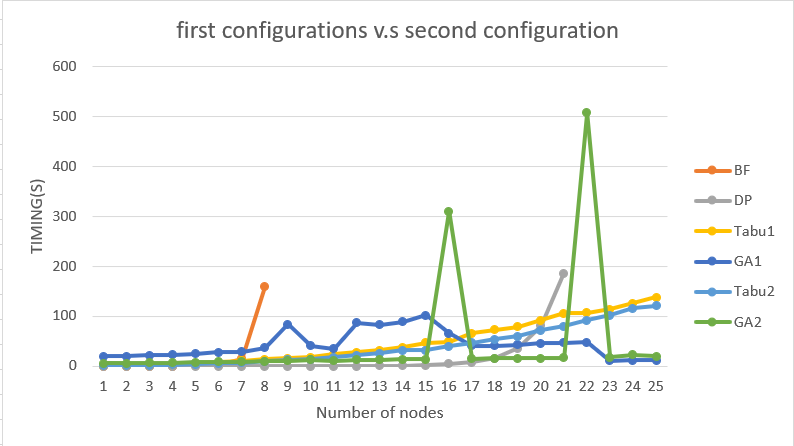
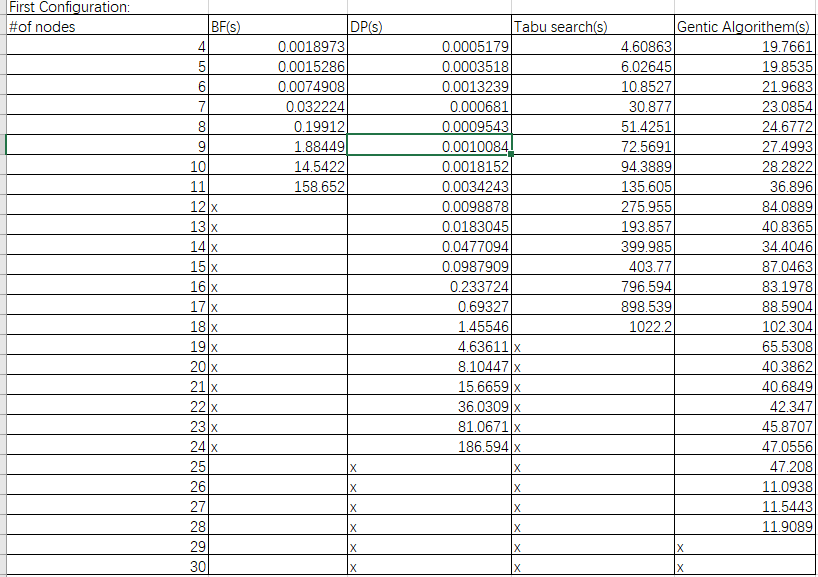
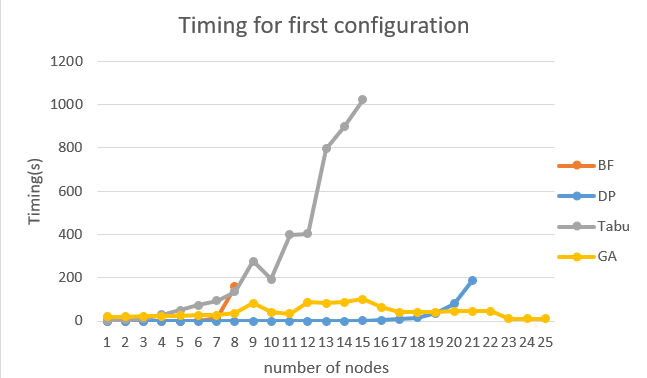
**REPORT4**

**TOTAL:**

****

**For the first configuration:**



****

**In Tabu search:**

1. Find neighbor strategy is :

swap(best\_path[p1], best\_path[p2])

1. The Tabu list size is 100

**In Genetic Algorithm:**

1.Fitness function: Always choose two minimum cost path from random generate

2.Selection: choose top two path by fitness function as the parents

3.Crossover: Assume crossover always happen and do the following to find the child

int temp = father[p];

if (mother[temp] != 1) {

swap(mother[temp - 1], mother[temp]);

}

return mother;

4.Mutation: Assume mutation happen in a random rate by the following rate generator:

int counter = 0;

int c = 0;

while (counter < 100) {

counter++;

int p = rand() % 100 + 1;

if (p == 2) {

c++;

}

}

// if 2 happen 5 times then mutation happen

if (c > 5) {

random\_child = path\_generator(path);

GA\_cost = distance(graph, random\_child);

my\_parents.push\_back(make\_pair(random\_child, GA\_cost));

if (GA\_cost < the\_min\_GA) {

GA\_best[0] = make\_pair(random\_child, GA\_cost);

}

else {

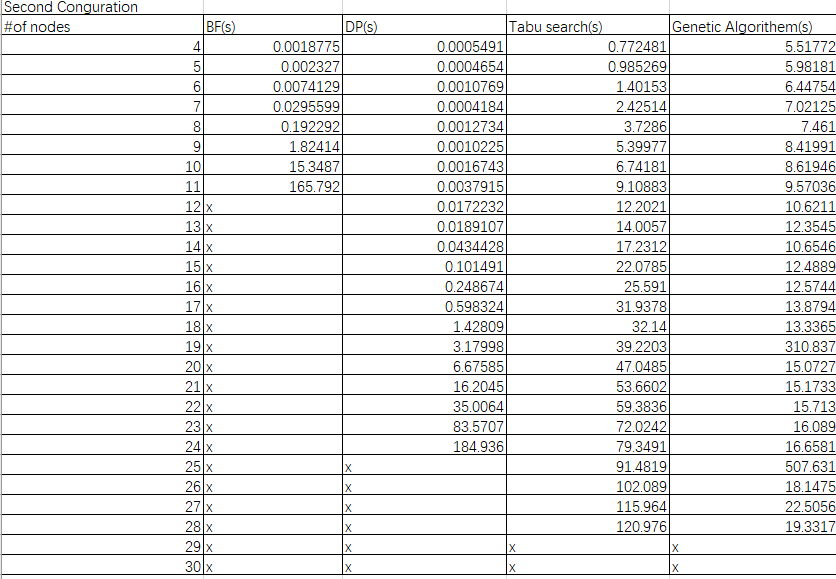
GA\_best[0] = make\_pair(father, the\_min\_GA);

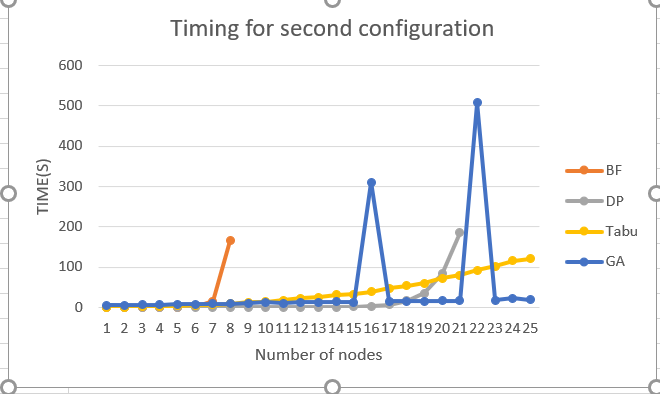
}

}

**Explanation for the first timing configuration:** For the actual timing, the BF and DP come smooth before certain nodes and increase sharply later. The BF method has a huge increase earlier than the DP method. On the other hand, unlike the BF method and DF, the Heuristic method, this time talking about Tabu search and Genetic algorithm, have oscillation timing line on the graph above. The Tabu method has a sharply increase when the nodes number are 12 but GA stay stable with little waving on the graph. The time complexity for Tabu and GA will be vary because different types of implementations. On my own first configuration, the time complexity for Tabu search is O(100000(n)\*(1000000)) = O() by double while with constant steps and two double for loops with variable n times to find possible neighbors and check that exist in Tabu list and a for loop for path generator in the first while loop. For my GA method, the time complexity is O(100000(n)+100000(n)) = O() by a while with a for loop to generate the first generations and another while with a for loop to crossover and select the child. Therefore, from the time complexity I achieved, it could explain why the actual time for the Tabu search increasing sharply before the GA search.

**For the second configuration:**





**In Tabu search:**

1. The Tabu list size increase two times to 200
2. The find neighbors strategy is the following：

// current swap with the pos = number in path into the new pos

int temp = best\_path[p1];

if (temp < best\_path.size() - 1) {

swap(best\_path[p1], best\_path[temp]);

}

**In Genetic Algorithm:**

1.Fitness function: Always choose two minimum cost path from random generate

2.Selection: choose top two path by fitness function as the parents

3.Crossover: Assume crossover always happen and do the following to find the child:

int temp = mother[p];

if (father[temp] != 1) {

swap(father[temp - 1], father[temp]);

}

return father;

4.Mutation: Increasing the rate happen in the mutation and assume mutation happen in a random rate by the following rate generator:

// assume mutation happen by random rate

// random rate generator

int counter = 0;

int c = 0;

while (counter < 100) {

counter++;

int p = rand() % 10 + 1;

if (p == 2) {

c++;

}

}

// if 2 happen 5 times then mutation happen

if (c > 5) {

random\_child = path\_generator(path);

GA\_cost = distance(graph, random\_child);

my\_parents.push\_back(make\_pair(random\_child, GA\_cost));

if (GA\_cost < the\_min\_GA) {

GA\_best[0] = make\_pair(random\_child, GA\_cost);

//cout << "here3";

}

else {

GA\_best[0] = make\_pair(father, the\_min\_GA);

//cout << "here4";

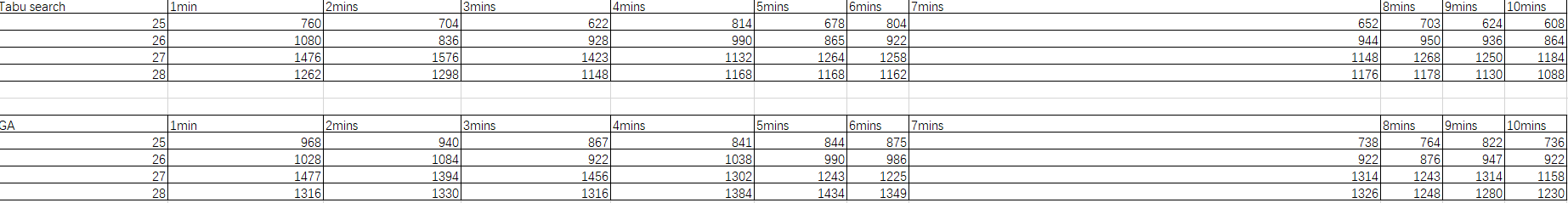
}

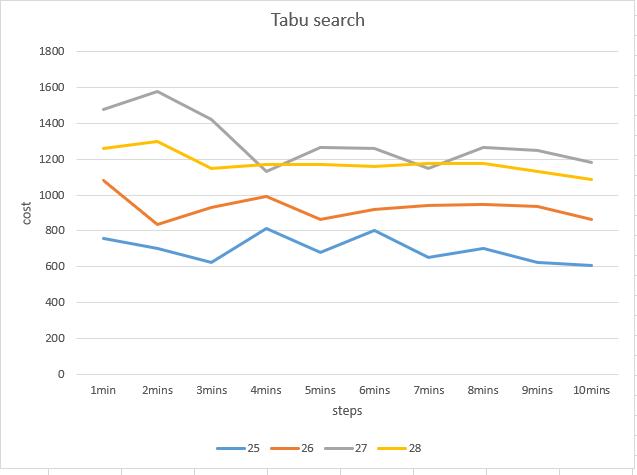
}

**Explanation for the second configuration:** For the actual timing, the BF and DP come smooth before certain nodes and increase sharply later. The BF method has a huge increase earlier than the DP method. On the other hand, unlike the BF method and DF, the Heuristic method, this time talking about Tabu search and Genetic algorithm, have oscillation timing line on the graph above. The Tabu method has a smooth increase which compare to the BF and DP but GA stay stable with big changes on the graph. One of the reason for the big changes on the graph is because the increasing mutation rate in the second configuration. In my own case, the increasing mutation rate bring the solution worse by increasing a huge amount of time with low efficiency to find a shorter path.The time complexity for Tabu and GA will be vary because different types of implementations. On my own second configuration, the time complexity for Tabu search is O(10000(n)\*(10000)) = O() by double while with constant steps and two double for loops with variable n times to find possible neighbors and check that exist in Tabu list and a for loop for path generator in the first while loop. For my GA method, the time complexity is O(100000(n)+100000(n)) = O() by a while with a for loop to generate the first generations and another while with a for loop to crossover and select the child.

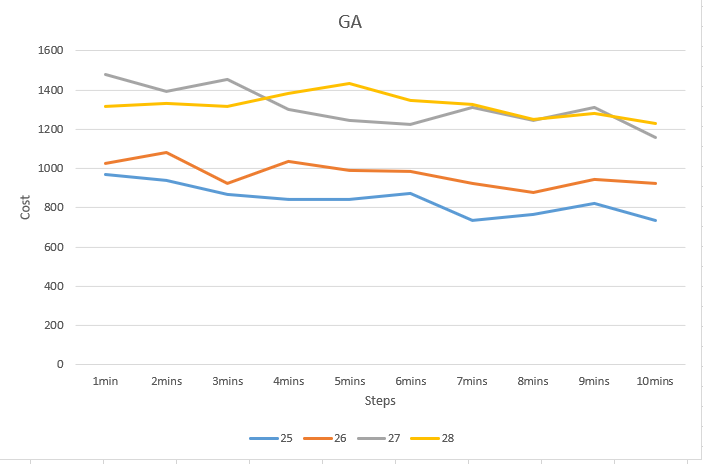
Therefore, comparing to the timing graph above and except the mutation, it is possible to explain why the GA have a more smooth graph than Tabu.

**Learning curve:**

****

****

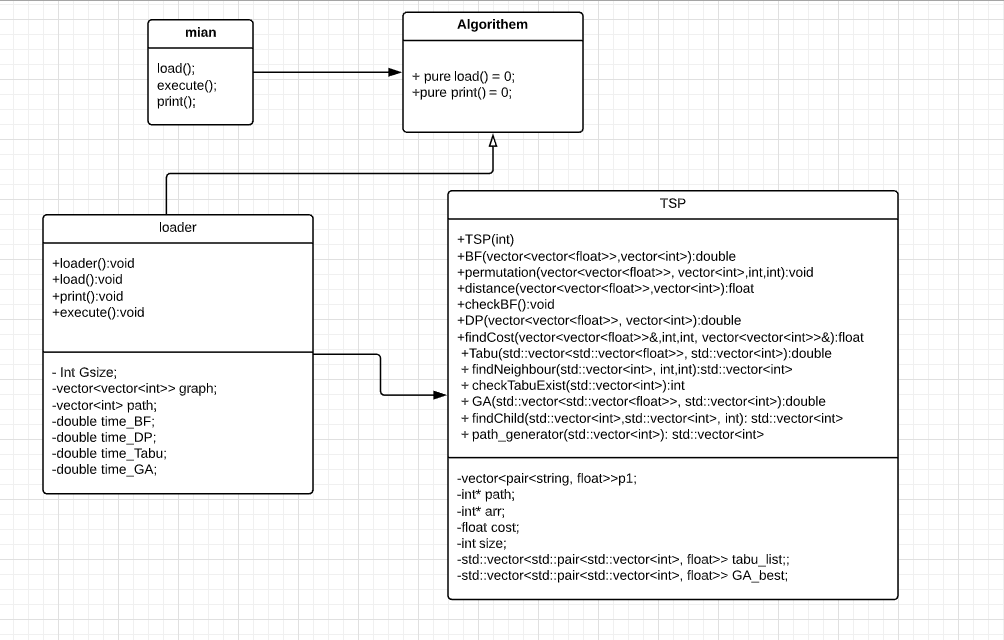
**Learning curve explanation for Tabu search:** The overview of the learning curve of Tabu search is when the running time grow up which means that the algorithm run more times in the long minutes time version can produce a lower cost path than shorter running time. In the Tabu search graph I got above, the nodes larger than lab3 I got are 25 to 28. And the curves for a smaller size of nodes have lower cost representing in the graph other than when the nodes size is 27. The reason for that is because when I make examples for position of nodes in 27, I put small value as x value and put a big value as y value compare to the node 25, 26 and 28. Even though the value make the curve(the number of nodes are 27) above the 28, the trend is similar to other curves. Also, because the Tabu start with a randomly picking path, the result I got will have oscillation in running the long time period and short time period. But running a whole algorithm in 10 mins have an absolutely improved performance than running in 1 or 2 minutes.

****

**Learning curve explanation for Genetic Algorithem:** The overview of the learning curve of GA is when the running time grow up which means that the algorithm run more times in the long minutes time version can produce a lower cost path than shorter running time. In the GA graph I got above, the nodes larger than lab3 I got are 25 to 28. And the curves for a smaller size of nodes have lower cost representing in the graph other than when the nodes size is 27. The reason for that is because when I make examples for position of nodes in 27, I put small value as x value and put a big value as y value compare to the node 25, 26 and 28. Even though the value make the curve(the number of nodes are 27) above the 28, the trend is similar to the other curves. Also, because the GA start with randomly picking paths and following with fitness function directly, each time the GA will have two potential paths to calculate the result instead of only one path like Tabu search. It will make GA more flexible to search around and more possibility to probably get into local minimum which will provide more costly results at the end. Therefore, the results I got will have oscillation in running the long time period and short time period. But running a whole algorithm in 10 mins have an absolutely improved performance than running in 1 or 2 minutes.

**For the different changes in Tabu search and Genetic algorithm to affect the results:** I increase the size of Tabu list and the running cycles and steps to obtain a more accurately and long time spending result. Also, the difference of neighbors finding strategy will affect the results in variable ways. The waving of the graph can explain that. Because the selected paths are based on keeping the current minimum and store into Tabu list, it is efficient to avoid the repeating paths. However, when the Tabu list is small, the list have high change to be rewrite from 0 position to upper positions. In this case, there is chance that a better solution was rewritten by new current minimum come form the neighbors. For the GA, the mutation rate is a highlight point in the whole functionality. In my own case, I decide to decrease the mutation rate in GA because when the mutation happen, the paths will jump to a less great direction and the cost increasing sharply when I did the recording of my data.

**Project Architecture Design:**



From the above UML diagram, I use strategy pattern to model my code like Lab3. In main.cpp, I only need to create a loader object and call all the necessary functions in an elegant way. It avoids me to put amount of codes in the main which let me understand clearly and easy to reimplement later. I also create two pure virtual function in Algorithm.h ,using pure virtual is a way to remind me overwrite the “load” and “print” function in the loader (which will inheritance from the Algorithm class) and in the case, I can put output system and loader system in a single interface. Loader is the class that I will direct create a loader object in main file and it like a connection connect the functional class and the main class. If I need to add any new functions in TSP class, I can simply call the function in the execute() in loader. The style of TSP class is easy to reimplement any new function or algorithms. The loader, at the same time, including the TSP class, which can call the TSP function in loader and all the function in TSP can share the same load and print function in loader because all the function in TSP need to print their results and read the input file. In conclusion, the strategy pattern can cut my whole program into small pieces, each piece has own utilization difference from others. It avoids me to put ton of codes into same class which is not easy to reread and reimplement.