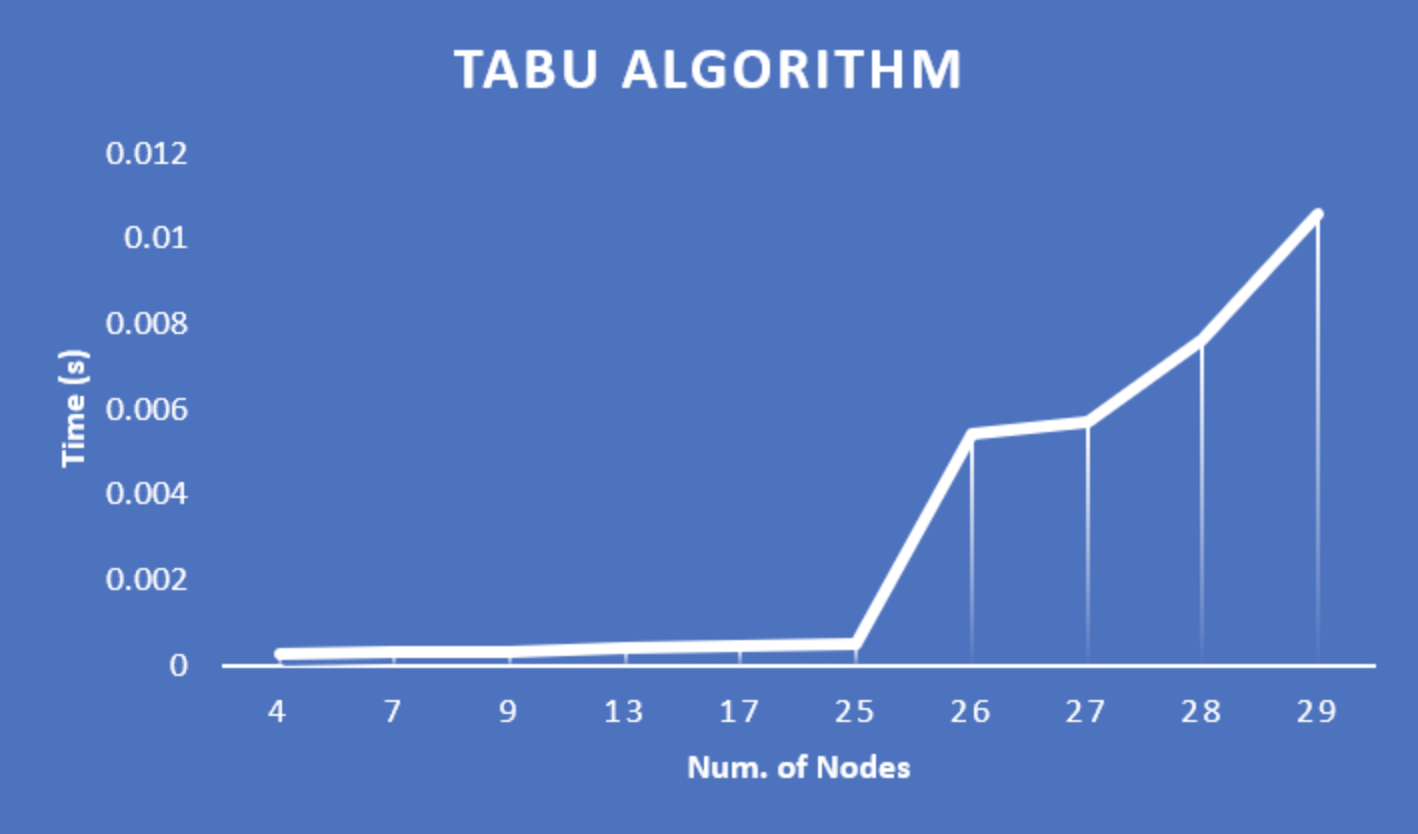
**Lab 4 report**

**TABU**

To implement Tabu search we have multiple techniques for neighborhood identification and the size of the list. I have explained the implementation techniques in below text:

One way of doing our task to identify neighborhood for a given solution is to select an index randomly in an array and after that select the previous element or the next element to exchange and go on for the next generation. Repeat this this step until the set size reaches its desired set value. Other wat to do the task for a given solution is to pick the first element of the array and again pick random index element from the array. After picking up these two elements, exchange them and move up one element from the first element and continue the process until it reaches the desired neighborhood size. Last method I used was to take the first element and fifth element from the array and exchange the values. Increase the indexes one by one and continue doing same process until desired neighborhood size is achieved.

We can change the size of the list as suitable for the graph entered by the user. You can observer that if you grow the size of the graph then it will do better and better in terms of time complexity. I observed that the second approach was the most beneficial in terms of results for larger sizes of graphs. With lesser number of nodes, you can sometimes get almost equal performance in all algorithms.



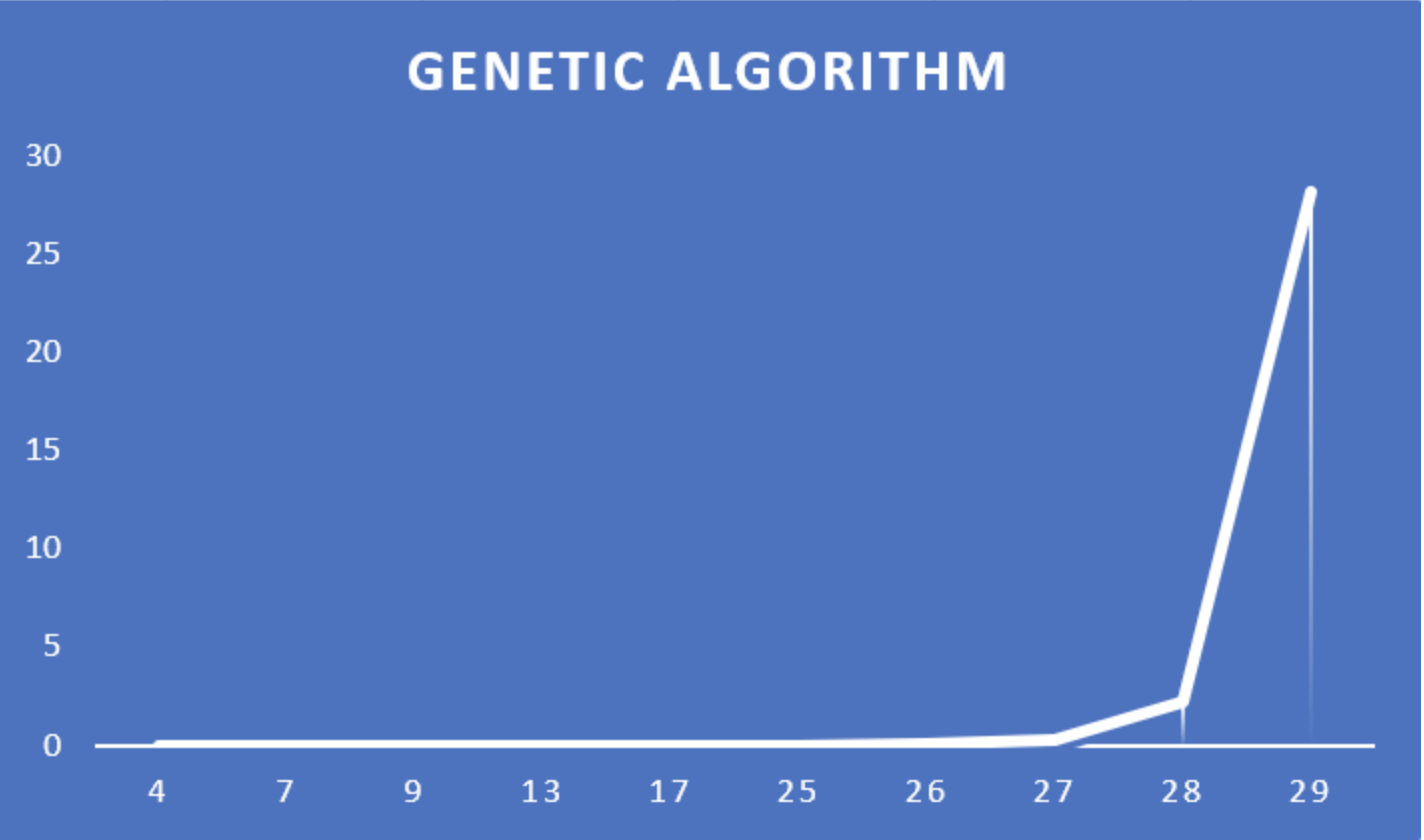
**GENETIC ALGORITHM**

In genetic algorithm technique we can certain configurations to do better search like tuning algorithm parameters like selection criteria, crossover rate and type, mutation rate and type. I have discussed my approaches that I have tried in my code:

To get the population I have selected 40% of it should be coming from selection and 60% of it should come from crossover. I have seen many methods for selection criteria but I have tried only a few among which I found elite chromosomes from the population and include them to have it in next generation. Second one I tried is two include every alternate one to have it in population and include them to have in next generation.

To implement crossover, I have generated subsets of parent population with 0.3 of the population parent size. This is known as two-point crossover. Two crossover point are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent.

To implement mutation strategy in my genetic algorithm I have selected elements randomly and then exchanged them. The mutation doesn’t happen every time so I have decided a variable to set mutation rate to change the rate of mutation in the population.

In the end, the genetic algorithm is way faster than the previous dynamic algorithm we have tried in lab3 but Tabu is still doing better than genetic algorithm in the search of optimal path in tsp. 

**GA vs Tabu**

GAs spend a lot of time sampling points that are trivially suboptimal. Suppose you're optimizing a function that looks like a couple of camel humps. GAs will dump points all over the place initially, and slowly converge to the points being at the top of the humps. However, even a very simple local search algorithm can take a point that the GA generates on the slope of a hump and push it straight to the top of the hump essentially immediately. If you let every point the GA generates go through this simple local optimization, then you end up with a GA searching only the space of local optima, which generally will greatly improve your chances of finding the best solutions. The problem is that when you start on real problems instead of camel humps, simple local search algorithms often aren't powerful enough to find the really good local optima, but something like tabu search can be used in their place.

There are two drawbacks. One, each generation of the GA goes much more slowly (but you need many fewer generations usually). Two, you lose some diversity, which can cause you to converge to a suboptimal solution more often. In practice, you should always include some form of local search inside a GA whenever possible.

**Design Patter and UML**

I have used factory pattern for the implementation of both the algorithms (GA and Tabu search). I selected this pattern because it works well with the algorithms that we have implemented in this lab. The managing of this pattern is super easy in my case and also different interfaces can be managed easily using this pattern. In my code AlgoHandler.h is one of the factories that I have developed in the game. This factory constructs a Tabu or Genetic algorithm object. My factory includes several smaller modules as well such as File reader. There is a graph class which helps creating Graphs with the help of input files with information of edges and weights. Interface for TSP is inherited by Algorithms.h which is further used to implement various algorithms needed for the search and selection.

