Farid Rajabi Nia fr2md lab: 11am

Prelab space and Time complexity:

In my prelab, I am using struct that hold and array of relationships, a string name and an integer to hold the degree. Then I am using a queue of pointers to hold them as one step before the actual sorting take place. Then, by popping the minimums from the queue we have our topological sort done. I have 4 loops in place, one for inserting into a queue that goes over each element if their degree is zero. Since insertion is constant time then we have linear running time of input size. Then we have A while loop and a for loop nested within which is two linear running time embedded. In this nested loop, we have front push and pop from the queue and all of them are constant time so they do not add no significant contributions to the total running time of this function. Therefore, it is going to be n^2 in the case of running time, n is our input size. In the main, there a loop and a push function that has linear running time depending on the input size which is the number of strings read from the file.

Inlab space and Time complexity:

The data structure used in travelling is mostly a string, a pointer to a memory address and a vector of a string including the name of cities. I believe that the vector holding the number of inputs which are the name of the cities. If we have 32 cities listed, then we have 32 *8 bytes per string.

in terms of running time complexity, I am looking for a shortest distance from one city to all unvisited cities so the I have go through n! different paths to find the shortest one. (n is the size of input i.e. number of cities in this case.) I have 3 loops in my file, but since N! is so gigantic and those linear loops are nothing in compared to the giant factorial running time. It is easy to implement but very inefficient.

Acceleration techniques to speed up the traveling salesperson problem:

Although TSP is a very simple problem to state and solve, the solution is astonishingly long as the input size gets larger. As I mentioned, its running time as defined above is n! and when n gets to 50, we are looking at years of computing power to solve the problem. Hence, we are looking for some techniques to speed up the process.

Effective use of GPU:

Graphic processor units are designed in a way that their multi-core processors have the ability to perform parallel data computation. Comparing to CPU, "the GPU performance is better on data parallel, throughput-oriented applications with intense arithmetic operations." Each GPU processor performs the same computation on a different piece of data provided. Paralleled processing capabilities of GPUs accelerate the computation speeds of these simple calculations for each iteration. Although, it is not solving the problem, but it helps reducing the processing time. An example of these GPUs studied in this research is CUDA, "a parallel computing platform and programming model introduced by NVIDIA." One of their findings is that by using GPU, they have managed to perform the task 14 times faster when dealing with 4000 nodes.

Local search algorithm:

Nearest Neighbor (NN) method is another approach to expediate the process. In this method, a random node is selected and then the route starts rom that node to its closest neighbor which has not been visited yet. This approach decreases the number of iterations and hence decreases the running time of the algorithm. The result of this study is as following “NN speeds up the algorithm 20.4% in the 9000-node instance whereas the speed-up is only 4.4% for the 500-node instance. ” Using Parallel computation and NN can probably help to improve the running time of TSP.

(https://ac.els-cdn.com/S2352146517301461/1-s2.0-S2352146517301461-main.pdf?_tid=1c6eb047-a53d-4254-b3ef-c45cc2852ef8&acdnat=1524701939_770fd1703c76cf04adc96efb15b04a09)

Inspirations from nature:

I was astonished by a number of studies on the subject that helped to improve the running time of TSP by using preexisted patterns in nature. For example, I have read an article in which “Dawson and Stewart (2014) applied the first parallel implementation of Ant Colony Optimization based edge detection algorithm on GPU.” The same observation from bees and birds also have led to many other optimizations. However, there was another article claiming, “Solving the Traveling Salesman’s Problem Using the African Buffalo Optimization.” It is mind blowing that by observing animal activities scientists can improve abstract problems like TSP. Scientists have created mathematical models based on the unique exploration and exploitation of buffalos in search of food. They cover the whole search space by “regular communication, cooperation, and good memory of its previous personal exploits as well as tapping from the herd’s collective exploits.”

The way that this algorithm works is that the buffalos are first initialized within the herd population. Then they are made to search for the global goal by updating their locations as they follow the current best buffalo in the herd. “Each buffalo keeps track of its coordinates in the problem space which are associated with the best solution it has achieved so far. This value represents the best location of the buffalo in relation to the optimal solution.” The algorithm follows this pattern: at each step, “it tracks the dynamic location of each buffalo towards the and depending on where the emphasis is placed at a iteration. The speed of each animal is influenced by the learning parameters.”

(https://www.hindawi.com/journals/cin/2016/1510256/)