INTRODUCTION

The Travelling Salesman Problem (TSP) is one of the most far-famed of optimization algorithmic problems in computer science. It is about finding a Hamilton cycle in a graph (the Hamilton cycle is a path that starts and ends in the same vertex (cycle) and contains each vertex of graph exactly once), incurring the lowest cost (sum of values assigned to graph's edges).

Historically, TSP derives from XIX century and was defined by British and Irish mathematicians. The name of the problem is related to a popular form of its exemplification – a man, an itinerant vendor, who has to visit a certain number of cities and end his journey where he started. Crucial aspect of his travel is to minimize, depending on a story, travel-time or route's length.

TSP problem is regarded as NP-hard, which basically means, that there is no known polynomial-time algorithm solving this problem (and there might be none!).

There are many variations on TSP, like admission of vertices repetition or differentiation of distances from vertex A to B and from B to A (asymmetry).

NOTATION

By n or V or |V| we mean number of vertices in the graph. By E we mean number of edges in the graph. We assumed, that E always equals n^2.

BRUTE FORCE

The easiest, the most obvious and, simultaneously, expensive solution is generating all possible permutations of vertices' arrangement, calculating cost for each of them and, finally, choosing the permutation with the lowest cost. Generating all of the permutations requires O(n!) time and calculating cost for one permutation is O(n), so overall complexity of the brute force solution is estimated by O(n*n!). To make this algorithm concurrent, just make each thread work on some portion of the generated permutations. All portions (subsets of all permutations) should be roughly equal size and pairwise disjoint.

Another variation of this problem is - Traveling salesman has to visit each city **at least once (instead of exactly once)** and then return to the starting city. In order to find shortest paths in weighted graph, we used Floyd–Warshall algorithm with O(n^3) complexity. The main idea of this algorithm is that the shortest path between vertices M and S, which contain vertex P, is sum of the shortest paths from M to P and from P to S. The rest of the algorithm and the overall complexity is the same as in the previous solution. Same trick with Floyd-Warshall can be used in DP solution.

DP (dynamic_programming_tsp_solution)

Requires O(N^2 * 2^N) time and O(N * 2^N) space and works for both symmetric and asymmetric (directed and undirected) graphs.

Let tab[mask][last_visited_vertex] be the cost of the shortest Hamilton walk in the subgraph generated by vertices in mask, that ends with last_visited_vertex. After proper initialization of the tab, the values can be calculated using that formula:

- if tab[mask][last_visited_vertex] value is already known, return tab[mask][last_visited_vertex],
- else tab[mask][last_visited_vertex]=min value of [tab[mask xor 2^ last_visited_vertex][j]+edge_cost(j, last_visited_vertex)] for all suitable j, where j is the vertex visited directly before last_visited_vertex.

Example:

1. n=8, mask1=00101100, last_visited_vertex1=2.
2. That means that vertices {2,3,5} have been visited and the last one visited was vertex number 2.
3. tab[mask1][last_visited_vertex1]=min(tab[mask1 xor (1<< last_visited_vertex1)][3], tab[mask1 xor (1<< last_visited_vertex1)][5])= min(tab[00101000][3], tab[00101000][5]).
4. tab[00101000][3] and tab[00101000][5] we calculate using recursion.

Source and more in depth description of the algorithm can be found here: http://codeforces.com/blog/entry/337.

In this algorithm memory (not time) turns out to be the bottleneck. For 24 vertices, peak memory used is 3GB and 25 vertices would require around 7GB.

DP AND BRUTE FORCE SOULTIONS CREDIBILITY
Both BF and DP solutions have been checked (up to 10 vertices instances) by submitting the solutions to the uva online judge, problem:
http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=1437
Source code is available in 10496 – Collecting Beepers and 10496 – Collecting Beepers DP.

NEAREST NEIGHBOUR ALGORITHM

The nearest neighbour is an example of a greedy algorithm. In this case salesman starts from random vertex and repeatedly visits the nearest vertices until every vertex is visited. In our implementation we used set which is related to O(logn) insert/find complexity (because of its tree structure) to recognize the nearest vertices and adjacency matrix to implement graph. Matrix review requires O(n^2) time, so total complexity of nearest neighbour algorithm is O(n^2*logn). It has to be said, that n^2 solution obviously exists, but we didn't bother to use that one, since O(logn) factor hardly ever matters.

Repetitive nearest neighbour (**salesman tries all vertices as the starting point**) is O(Nearest neigbour) * **O(n)** = O(n^3logn) in this case.

Both, repetitive and unrepeatable versions, work for directed and undirected graphs for instances up to around 5000 vertices. Their peak memory used is 1,2GB for n=5000.

DOUBLE MST

It creates minimum spanning tree (using Prim's algorithm), then does n (each vertex as the starting point) times DFS (preorder traversal of the tree). Algorithm constructs for every DFS a tour, and finally it chooses the best tour. The output tour is guaranteed to be max not more than twice as long as the optimal solution (**only when triangle-inequality holds**). MST complexity is O(V^2), MST could be done in O(E * logV) but there is no point in doing that here, since we are considering only complete graphs. Algorithm works of course only for undirected graphs. DFS is O(V) and we use DFS |V| number of times, to find the best tour constructed by MST. Overall complexity is O(V^2). Algorithm should work easily for instances up to 10^4 number of vertices.

In our tests for n==5000 peak memory used is 300MB and for n==10k it is 1.14GB.

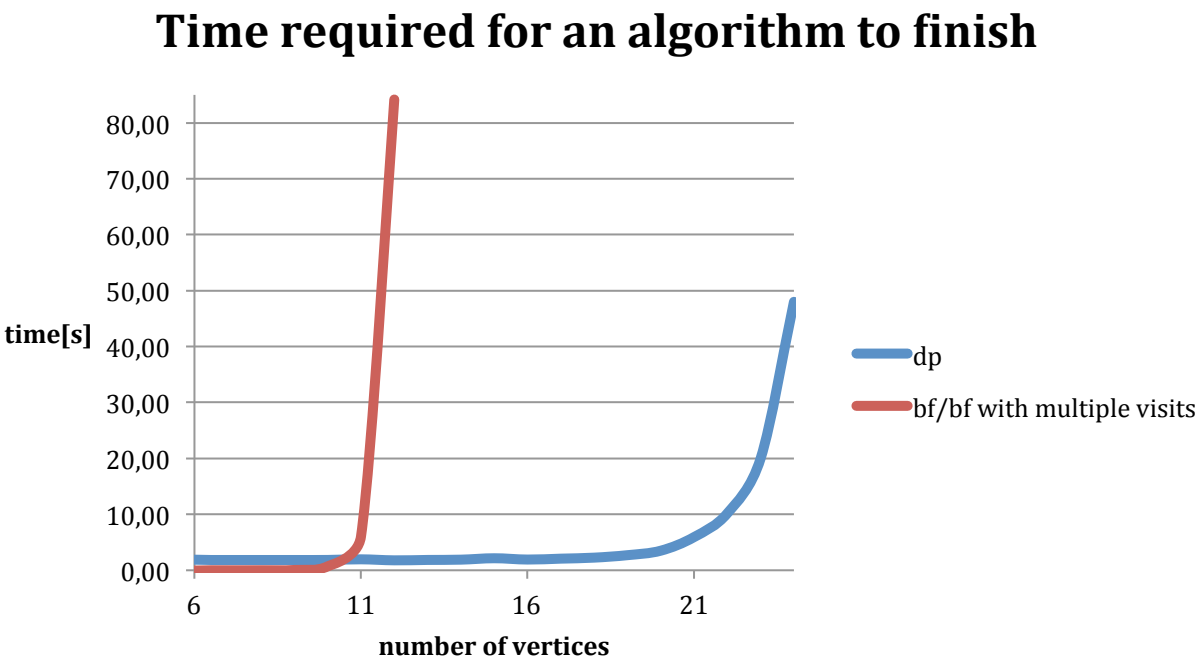
Important fact: "Without the **triangle inequality**, a polynomial time approximate algorithm with constant approximation ratio not exists unless P=NP." We leave the proof as an exercise for the reader.

SUMMARY

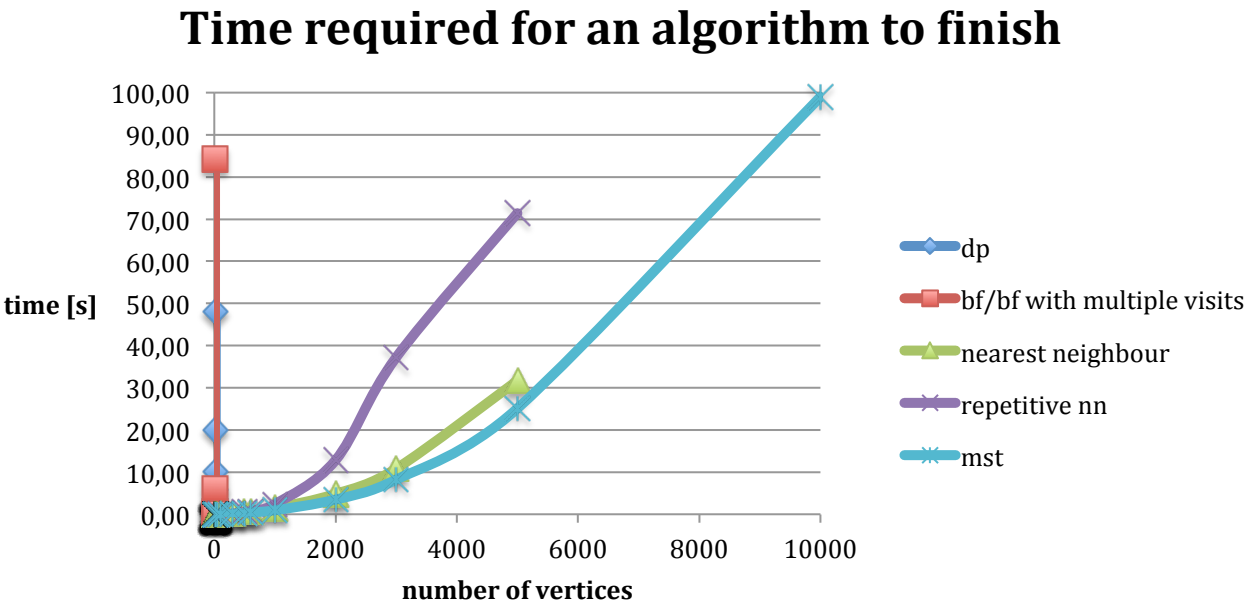
Reasonable choice of an algorithm ought to be based on a number of analyzed vertices and expected accuracy of the result.

If the exact result is required, one should choose between the dynamic programming and the brute force. The dynamic programming approach is much faster than the brute force for number of vertices greater than 10, and works for instances up to 24 vertices. For number of vertices > 24 a branch and bound

algorithm is required, which is beyond scope of this paper.



For huge graphs, time is the main issue, solutions always offering exact results are simply too slow. For instances with less than 5000 vertices it is advisable to try both repetitive nn and mst solutions, because it is rather unpredictable, which of them would work the most efficient for miscellaneous graphs. For graphs with more than 5000 vertices only minimal spanning tree algorithm will work.



Most of our implementations work only for integer edge costs, provided that the cost of the Hamilton cycle is less than 10^9 (because in our solutions infinity=1e9). You could easily modify them to work with double edge costs and longer cycles than 1e9.

Sample tests:

12cities_symmetric.txt

- optimal tour found by both dp and brute force: 1733,
- brute force multiple visits 1675,
- MST 2848,
- nearest neighbour 2110,
- repetitive nearest neighbour 1733 (optimal) but other vertices order than DP or BF algorithms.

11cities_symmetric.txt (no triangle-inequality has been proved by MST and optimal results)

- brute force 1675,
- brute force multiple visits 1366,
- dp 1675,
- MST 3431
- nearest neighbour 2285,
- repetitive nearest neighbour 2139.