REPORT ON ASSIGNMENT 2:
TRAVELING SALESMAN PROBLEM (TSP)

COURSE: ADVANCED ALGORITHMS

NICOLA ADAMI – FRANCESCO PENNA – SOLMAZ MOHAMMADI

INTRODUCTION

The **traveling salesman problem** (**TSP**) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?". In other words, given the coordinates x, y of N points on the plane (the vertices), and a weight function $w(u, v)$ defined for all pairs of points (the arcs), find the simple loop of minimum weight that visits all the N points. The weight function $w(u, v)$ is defined as the Euclidean or Geographic distance between the points u and v and it is symmetric and respects the triangular inequality. In the theory of computational complexity, TSP belongs to the class of **NP-complete** problems, meaning it is not possible or feasible to find an exact solution, meaning it is necessary to search for an approximate one. In this project, we implemented a **2-approximation algorithm** and **two constructive heuristics** to approximate the solution to this problem.

DATA STRUCTURE

GRAPH REPRESENTATION

The dataset provided for this project is a list of points with their coordinates in a 2-dimensional space. The files also contain some additional information such as the number of points and the type of coordinates which can be euclidean or geographic. The tsp graph is an undirected, connected graph. Therefore, for the representation of the TSP graph, each point's connectivity with all the points in the graph is considered. The points in the graph are represented with a class object Node which represents the name of the points, and the x, y coordinates. The distances between every two nodes are precomputed and represented with a python dictionary with a string of connected vertices u, v as the key and weight of the edge $w(u, v)$ as value.

DISTANCE COMPUTATION (WEIGHT FUNCTION)

We obtain the weights of the edges using two different weight functions (based on the type of the coordinates given in the problem). The weights are computed and saved in the data structure to boost the algorithms' efficiency.

GEOGRAPHIC DISTANCE

Given the longitude and latitude of a point, we convert them to radians according to the provided formula and compute the geographic distance between every two points.

EUCLIDEAN DISTANCE

The Euclidean distance of two points p and q in Euclidean n-space is obtained by the formula below:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

ALGORITHMS

THE 2-APPROXIMATE ALGORITHM BASED ON MST

The prim's algorithm is used to find the minimum spanning tree for a given graph. The main idea of this 2-approximation algorithm is to find the MST of the graph and find the Hamiltonian cycle using the preorder tree traverse method and creating a cycle.

IMPLEMENTATION CHOICES

We used the prim's algorithm implemented in the previous project with some modifications to meet this project's needs. We added the attribute *children,* a list to keep track of each node's children, to make the tree traverse possible. The prim's algorithm is designed to receive the TSP graph points, and edges, and make an MST graph, on which the algorithm can work. A random point is assigned as the root of the tree as the initialization, the prim's algorithm simply modifies the vertices of the MST graph and saves each node's value, parent, and children. By only giving the root of the tree to the tree traverse function $preorder(cycle, r)$, the cycle is created by saving visited nodes in order. To complete the cycle, we connect the last visited node to the root and compute the cycle cost. Notably, this scenario works only because the graph is connected, which is a property of the TSP graph.

COMPLEXITY

Prim's algorithm has the complexity $O(mlogn)$, and the preorder function is in order of $O(m + n)$. Therefore, the complexity of the 2-approximation algorithm is $max\{\, O(mlogn), O(m + n)\,\} = O(mlogn)$

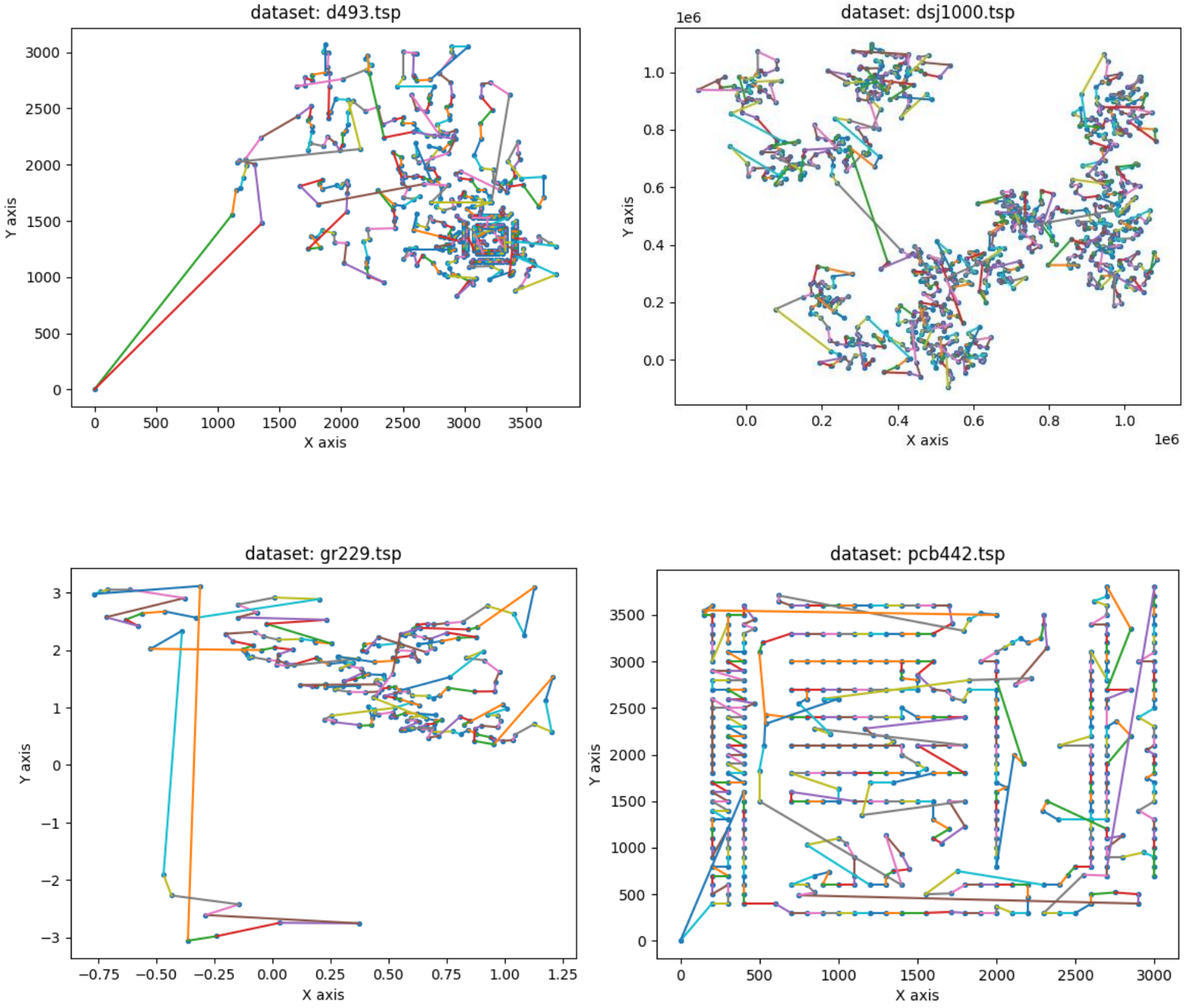


FIGURE 1. THE TSP SOLUTIONS OBTAINED BY THE 2-APPROXIMATION ALGORITHM

CONSTRUCTIVE HEURISTICS

NEAREST NEIGHBOR

The theoretical idea for this heuristic is that, starting from a **single-node path with zero cost**:

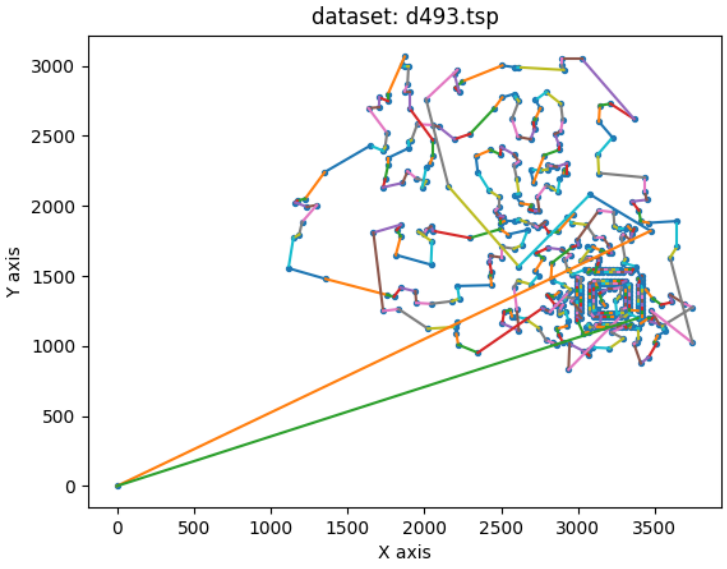
- Let $(v_1, \dots, v_k)$ be the current path. The nearest neighbor v_{k+1} is the vertex not in the path with a minimum distance from v_k. The v_{k+1} is inserted immediately after v_k.
- Repeat this process until all the nodes are included in the cycle.

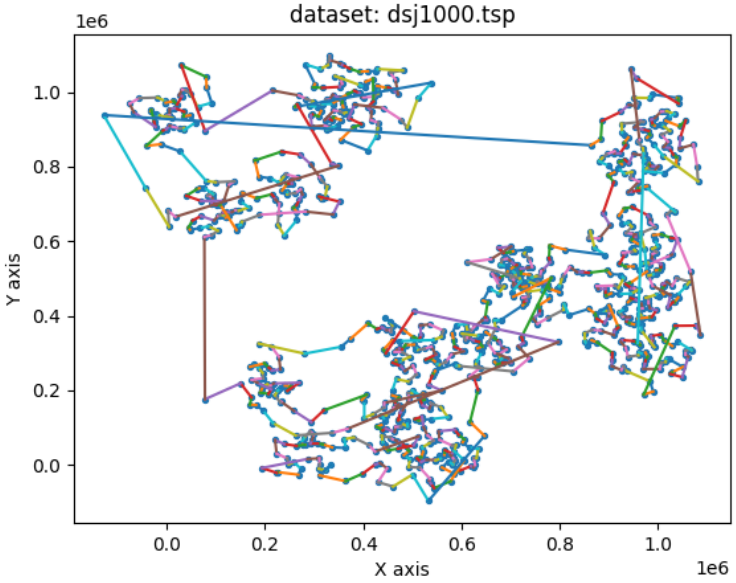
IMPLEMENTATION CHOICES

We choose the starting point **randomly** and build the cycle from that node. For efficiency, we separate the solution from the remaining points by creating two lists: (i) One with all the nodes that are yet to be processed and (ii) all the points included in the cycle.

COMPLEXITY

For finding the nearest neighbor of a vertex we loop over all its adjacent (*m times*), and we do this for every vertex (*n times*). Hence, the complexity is $O(m * n)$.





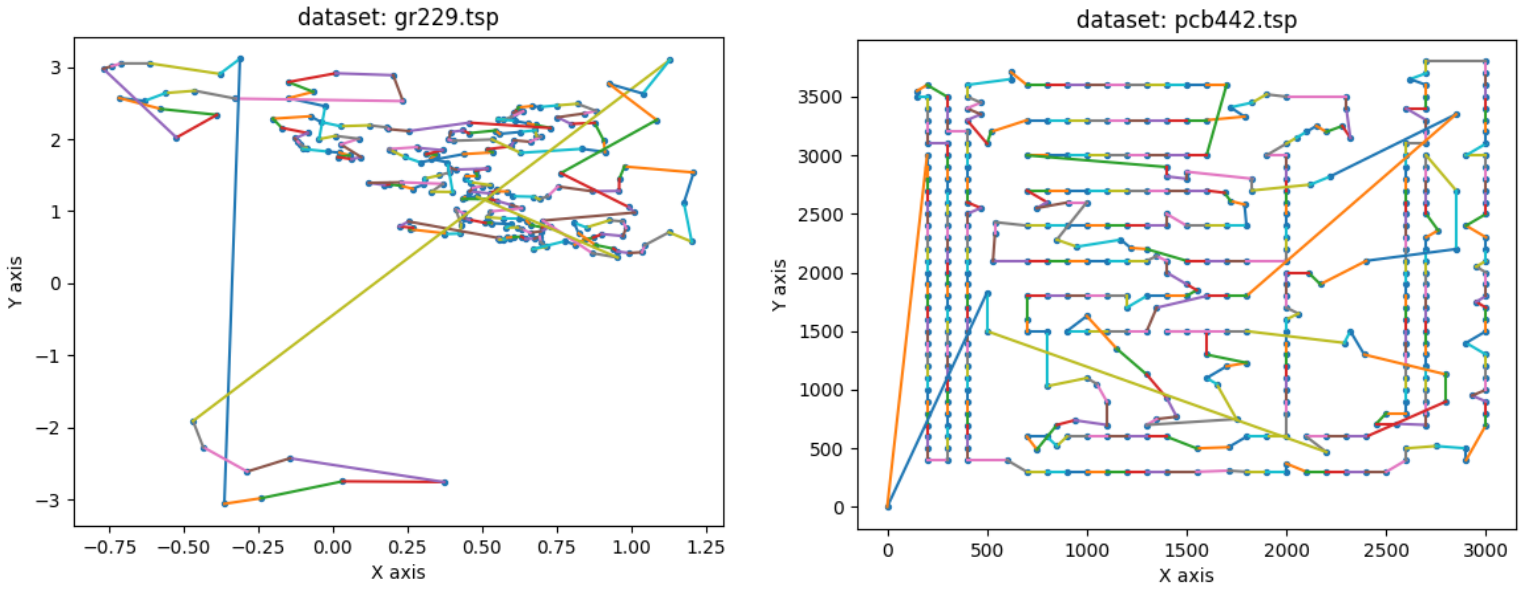


FIGURE 2. THE TSP SOLUTIONS OBTAINED BY THE NEAREST NEIGHBOR HEURISTIC

RANDOM INSERTION

Starting from the single-node path with a cost $of\ 0$, and building the partial circuit $(0, j)$ minimizes the cost $w(0, j)$:

- Randomly select a vertex k not in the circuit, and insert k into the partial circuit $\{i, j\}$ which minimizes the property $w(i, k) + w(k, j) - w(i, j)$
- Repeat this process until all nodes are included in the cycle

IMPLEMENTATION CHOICES

We choose the starting point randomly and build the initial partial circuit with that node. For efficiency, we separate the solution from the remaining points with creating two lists: (i) One with all the nodes that are yet to be processed and (ii) all the points included in the cycle. In order to be able to do the process of insertion efficiently, we build the cycle by adding strings of edge keys and access the partial circuits that build the cycle easily.

COMPLEXITY

Like nearest neighbor, for finding the minimizing edge for each vertex, we loop over all the edges in the path (*m-times*), and we do this for all the vertices *(n-time*). Therefore, the complexity of random insertion is also $O(m * n)$.

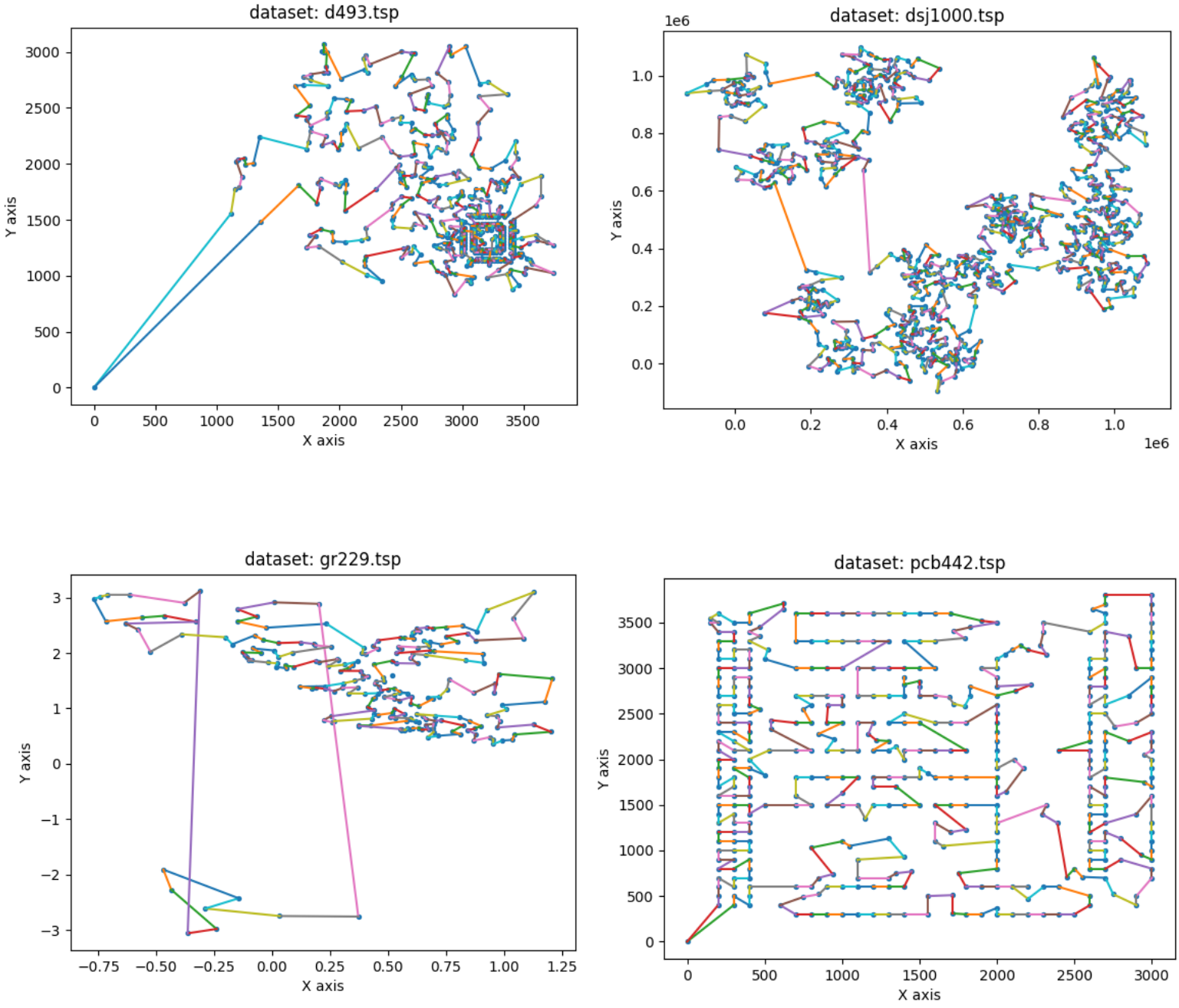


FIGURE 3. THE TSP SOLUTIONS OBTAINED BY THE **RANDOM INSERTION** HEURISTIC

RESULTS

The table below shows the results obtained by running the algorithms on the given datasets 100 times and 5 instances. In general, the random insertion has the lowest error rate among all algorithms and the nearest neighbor has the lowest computational time. (See figure.4)

INSTANCE	NEAREST NEIGHBOR			RANDOM INSERTION			2-APPROXIMATION		
	Solution	Time	Error	Solution	Time	Error	Solution	Time	Error
BERLIN52.TSP	8181	3687741	0.084	7591	4663976	0.006	9550	4458767	0.266
BURMA14.TSP	3827	387829	0.151	3327	463434	0.001	3688	408632	0.109
CH150.TSP	7713	29034118	0.089	6805	26529285	0.042	8635	35831578	0.322
D493.TSP	40915	2,92E+08	0.166	37294	3,84E+08	0.065	44997	3,58E+08	0.285
DSJ1000.TSP	22980721	1,31E+09	0.231	20312741	1,9E+09	0.088	25209654	1,87E+09	0.351
EIL51.TSP	482	5012627	0.131	437	4807654	0.025	551	3884370	0.293
GR202.TSP	48658	54839418	0.166	42946	66179521	0.046	50170	61025736	0.249
GR229.TSP	159399	68557466	0.184	138925	84351220	0.032	171922	76135516	0.277
KROA100.TSP	24698	12590144	0.160	21653	15434171	0.017	28648	14266972	0.346
KROD100.TSP	24852	12320025	0.167	21848	15303335	0.026	277742	14845681	0.302
PCB442.TSP	50229	2,5E+08	0.166	55895	3,41E+08	0.100	68996	3,11E+08	0.358
ULYSSES16.TSP	7927	453007	0.155	6894	541466	0.005	7772	484921	0.133
ULYSSES22.TSP	8158	758886	0.163	8253	925068	0.169	8210	853912	0.170

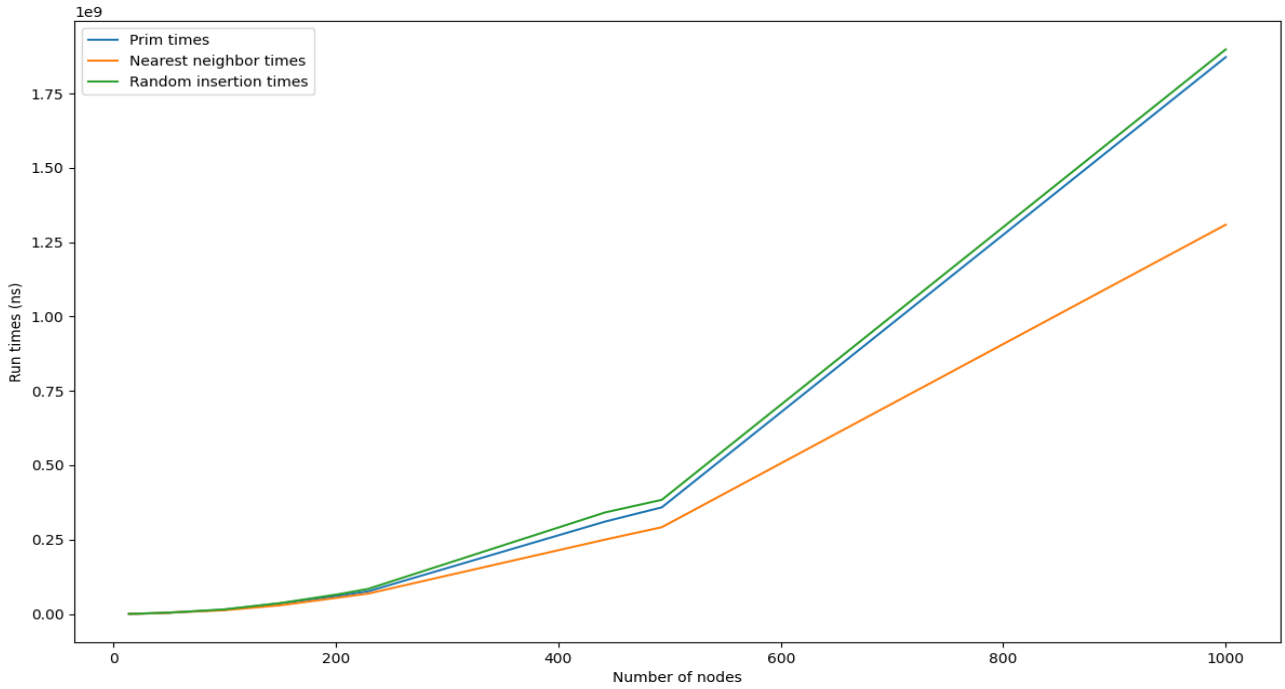


FIGURE 4. COMPARISON OF RUN-TIMES BETWEEN THE THREE ALGORITHMS

CONCLUSION

In the theoretical time complexities, the worst case for 2-approximation is better than the two heuristics. However, we observed that in practice the nearest neighbor has better performance (timewise) because of its simplicity. In the case of solution optimality, the random insertion has the lowest error rate among all three algorithms. We believe that this is due to the power of randomized algorithms and their flexibility when it comes to practical implementations. Moreover, all the error rates are not dramatically large, and all three algorithms have shown good performance in both aspects of time efficiency and accuracy of results.