Project Report

Introduction to Algorithm Engineering

Bharat Sahlot — Kyrylo Shyvam Kumar

1 Introduction

The paper[1] presents a data structure and relevant algorithms for fast t-distance queries. We implemented the algorithms for connected undirected weighted graphs represented using Adjacency List.

2 Data Structure

The data structure presented is called *Oracle*. It requires a pre-processing step. The pre-processing step does most of the work, therefore queries are really fast.

The data structures takes as input a connected undirected weighted graph G and an integer K.

2.1 Pre process Algorithm

According to the paper this step takes time $O(kmn^{\frac{1}{k}})$. The actual time complexity depends on the shortest path algorithm used. In our case we went with djikstra, therefore we have a slightly worse time complexity.

The algorithm selects vertices randomly with a probability of $\frac{1}{k}$. We used `std::bernoulli_distribution` for this. We used STL data structures like `std::vector`, `std::priority_queue` and `std::unordered_map` heavily. We also use `boost::heap::fibonacci_heap` for implementation using Fibonacci Heap.

2.2 Query Algorithm

This is a simple algorithm. It takes two vertices as input and returns the k-spanner distance between them.

3 Performance

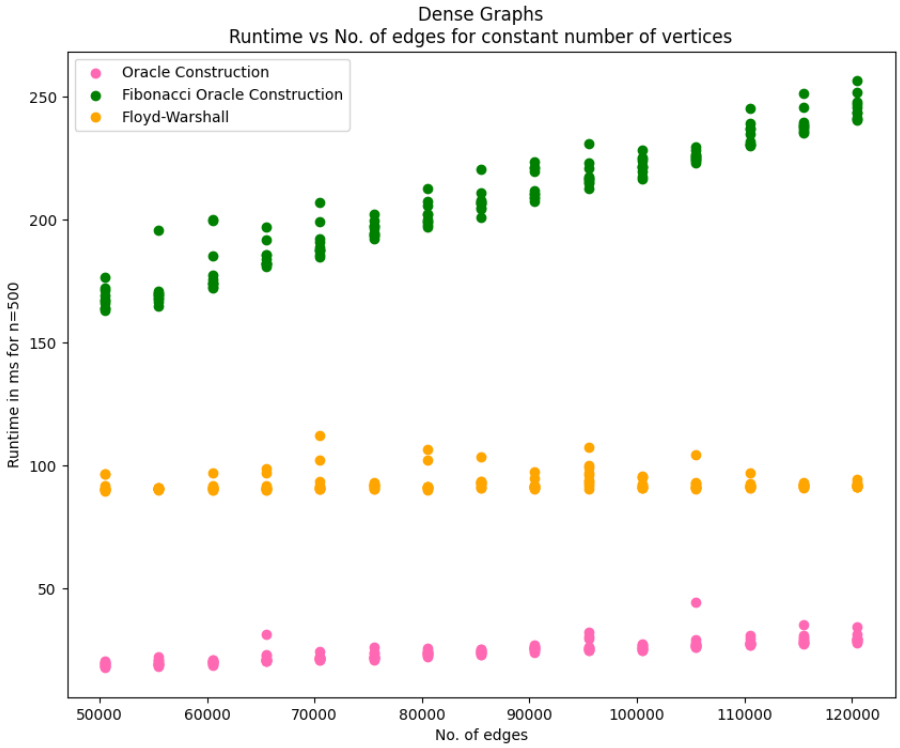
We have two implementations of the algorithm (one using `std::priority_queue` and other using `boost::heap::fibonacci_heap`), we compare them against Floyd-Warshall and Dijkstras algorithm.

3.1 Random Graphs

We generate random connected graphs of different vertices and edges, and we obtained the following performance graph. To account for varying nature of random graphs, around 10 were generated for given parameters to make sure data follows appropriate distribution.

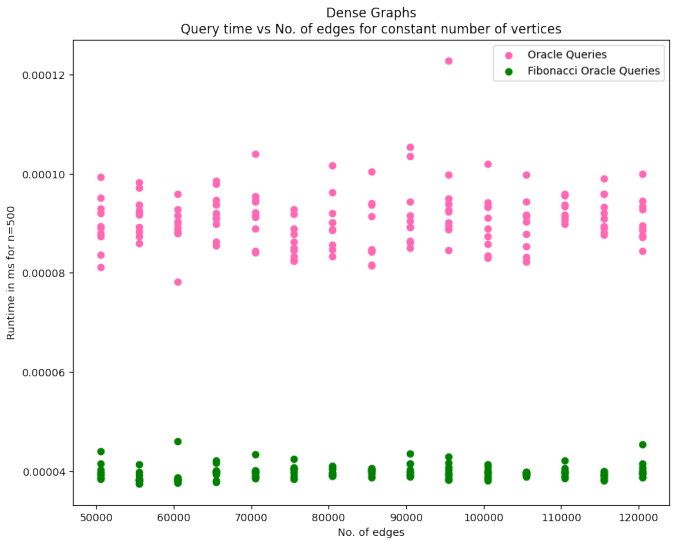
3.1.1 Varying number of edges while keeping number of vertices constant:

Figure-1



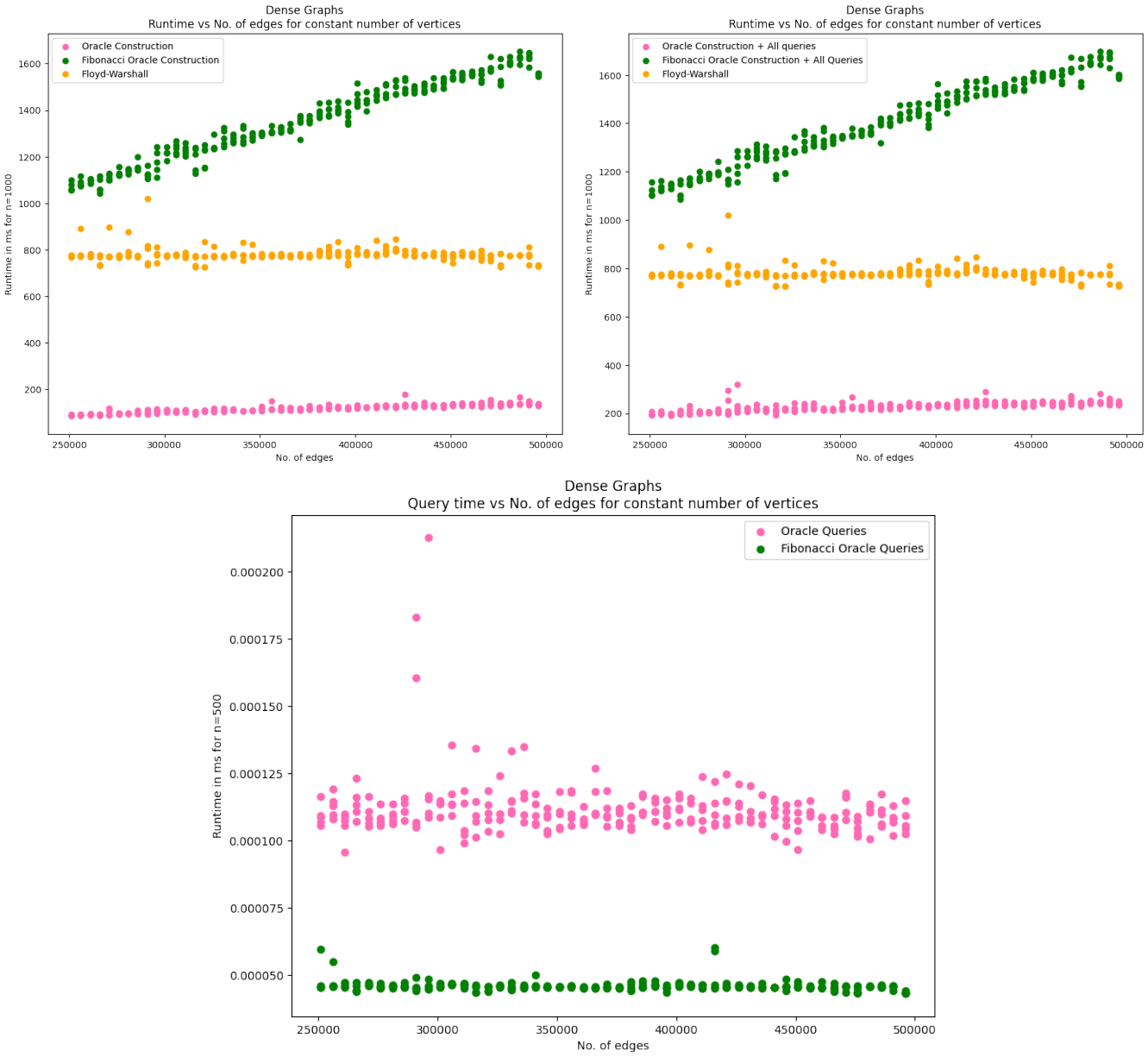
In the given Figure-1, it can be noticed that Floyd-Warshall's runtime remains same and Fibonacci Heap Oracle has increasing runtime. For Normal Oracle it is also increasing, with less visibility. The increase is because of theoretical complexity of $O(kmn^{\frac{1}{k}})$.

Figure-2 and Figure-3



The Figure-2 contains a similar as previous plot, but with accounting for querying time for all pairs of edges. The little shift of runtime of oracles can be observed. The next Figure compares the querying times of both Oracles. There is no increase as the number of edges increases, but the normal Oracle is a little worse due to difference in implementations.

Figure-3,4,5



These plots are similar to previous ones. The number of edges has been increased.

3.1.2 Varying number of vertices while keeping number of edges constant:

Figure-7

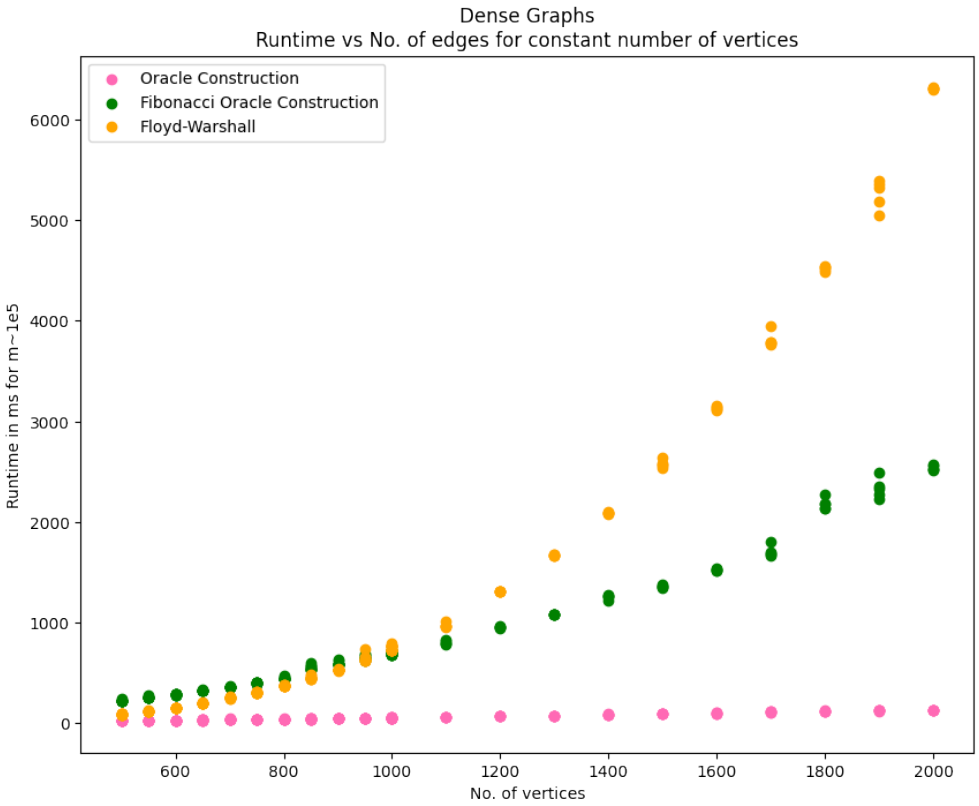
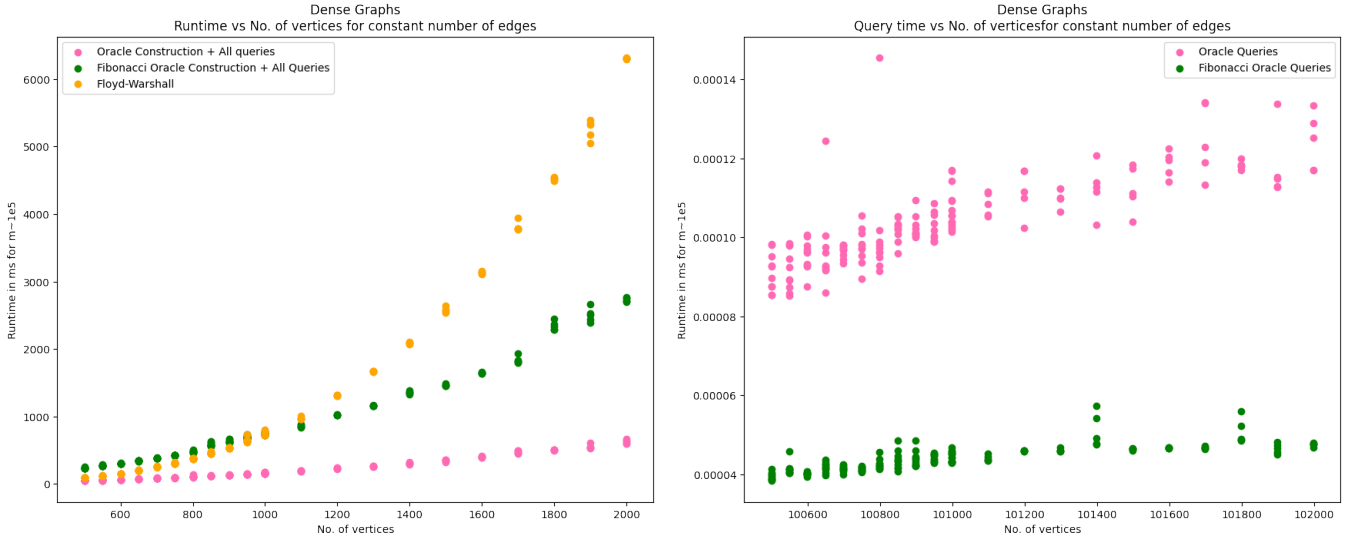


Figure-8,9

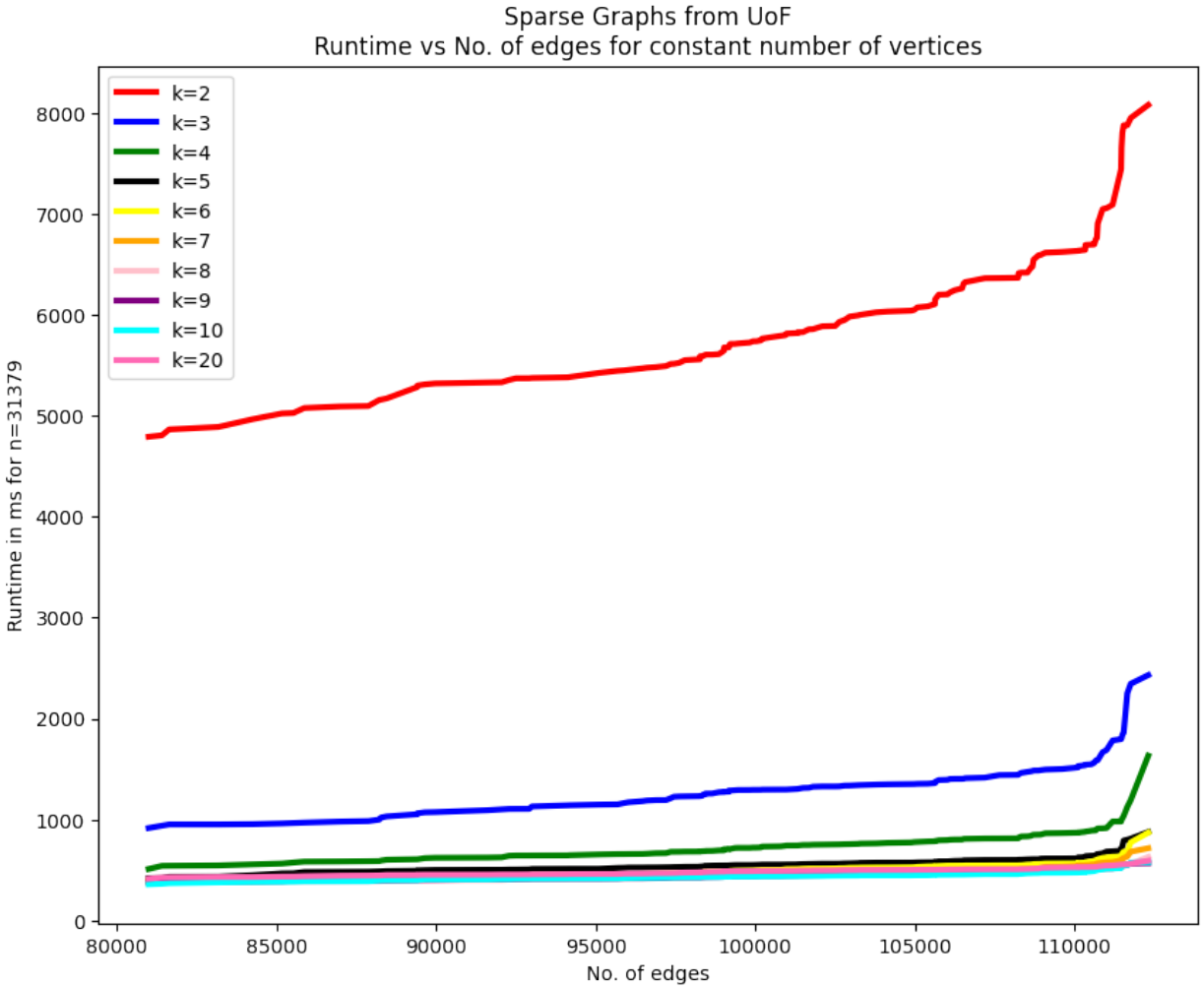


Here we see that runtime of Floyd-Warshall grows much faster than Oracles. The higher construction time for Oracle using Fibonacci Heap arises due to high constant factors of this data structure, but query time is unaffected and shows different trend.

3.2 Real World Graphs

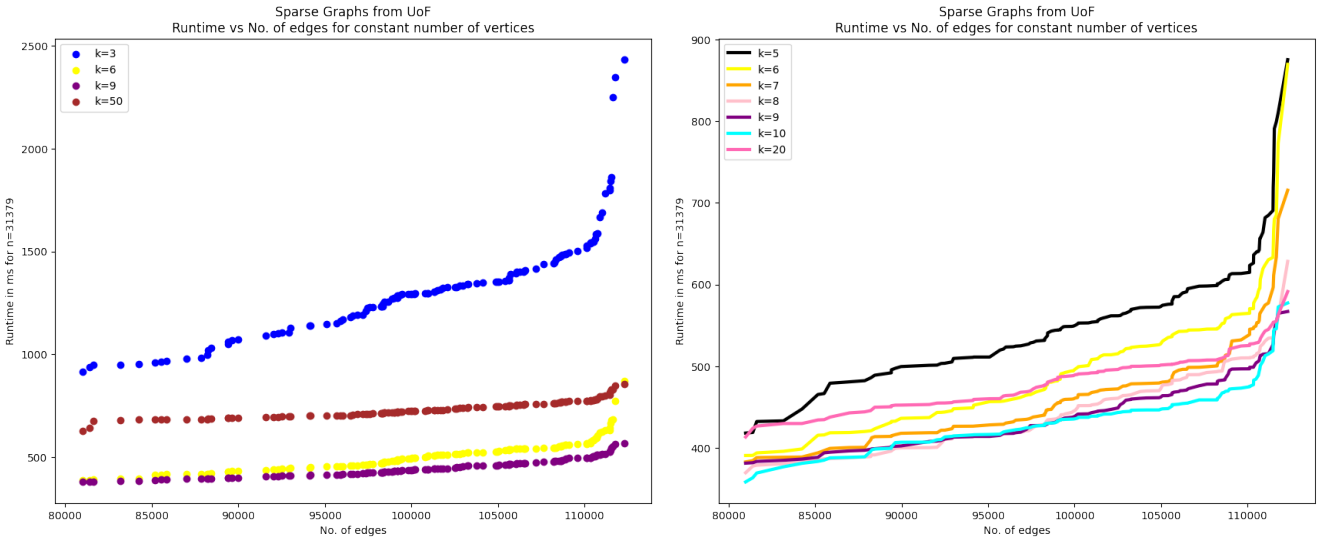
We used 122 CAIDA-AS graphs from UoF repository. These are weighted graphs (with weights 1,2,3,4) and we ensure that they are connected. Number of vertices stays constant at 31,379 and number of edges is around 100,000 (different for each graph). Each file contains a full AS graph derived from a set of RouteViews BGP table snapshots.
We got these real world graphs from the **Graph** directory on the server, and we obtained the following perform-aces from them.

Figure-10



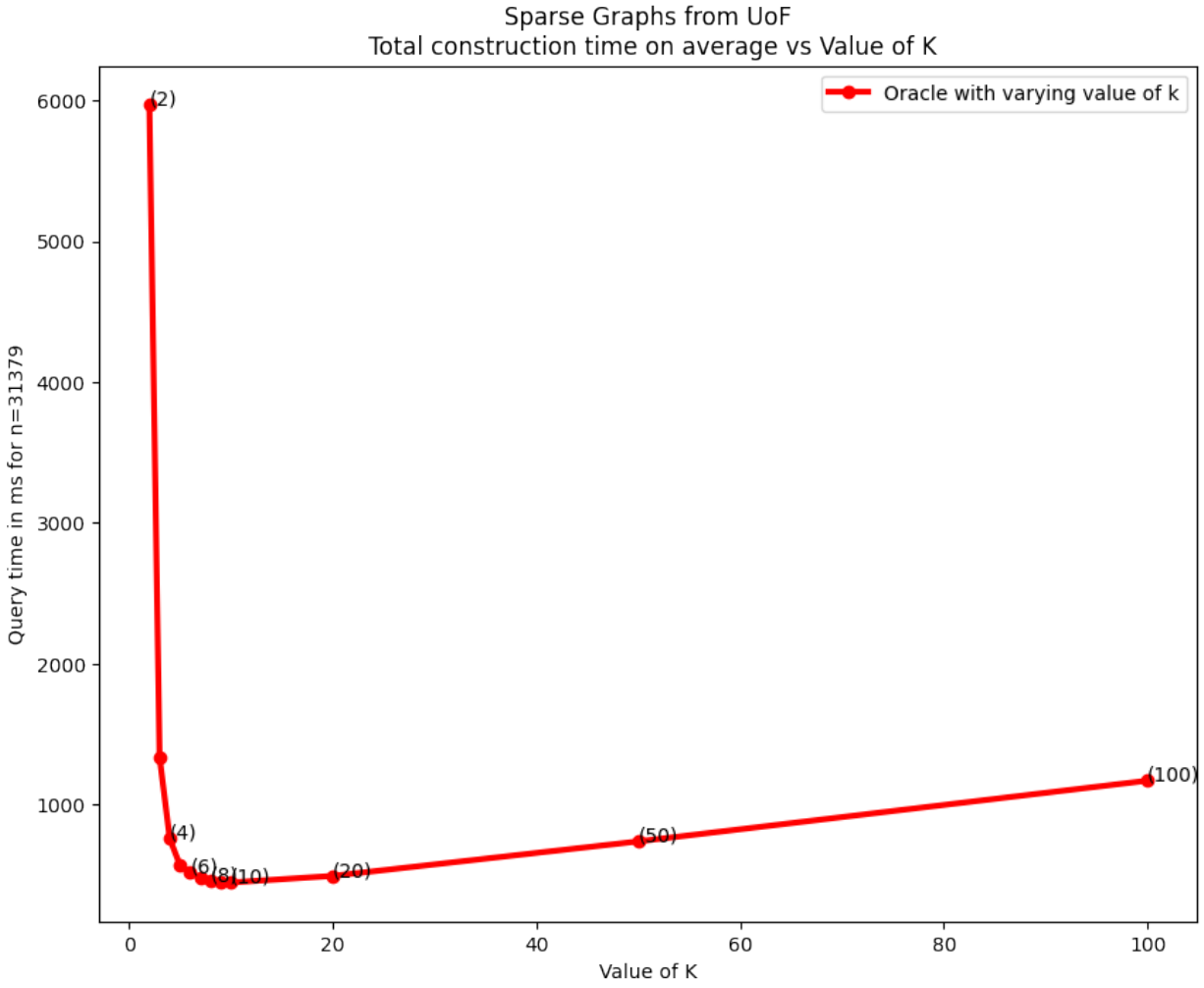
Following is a closer look on plot.

Figure-11,12



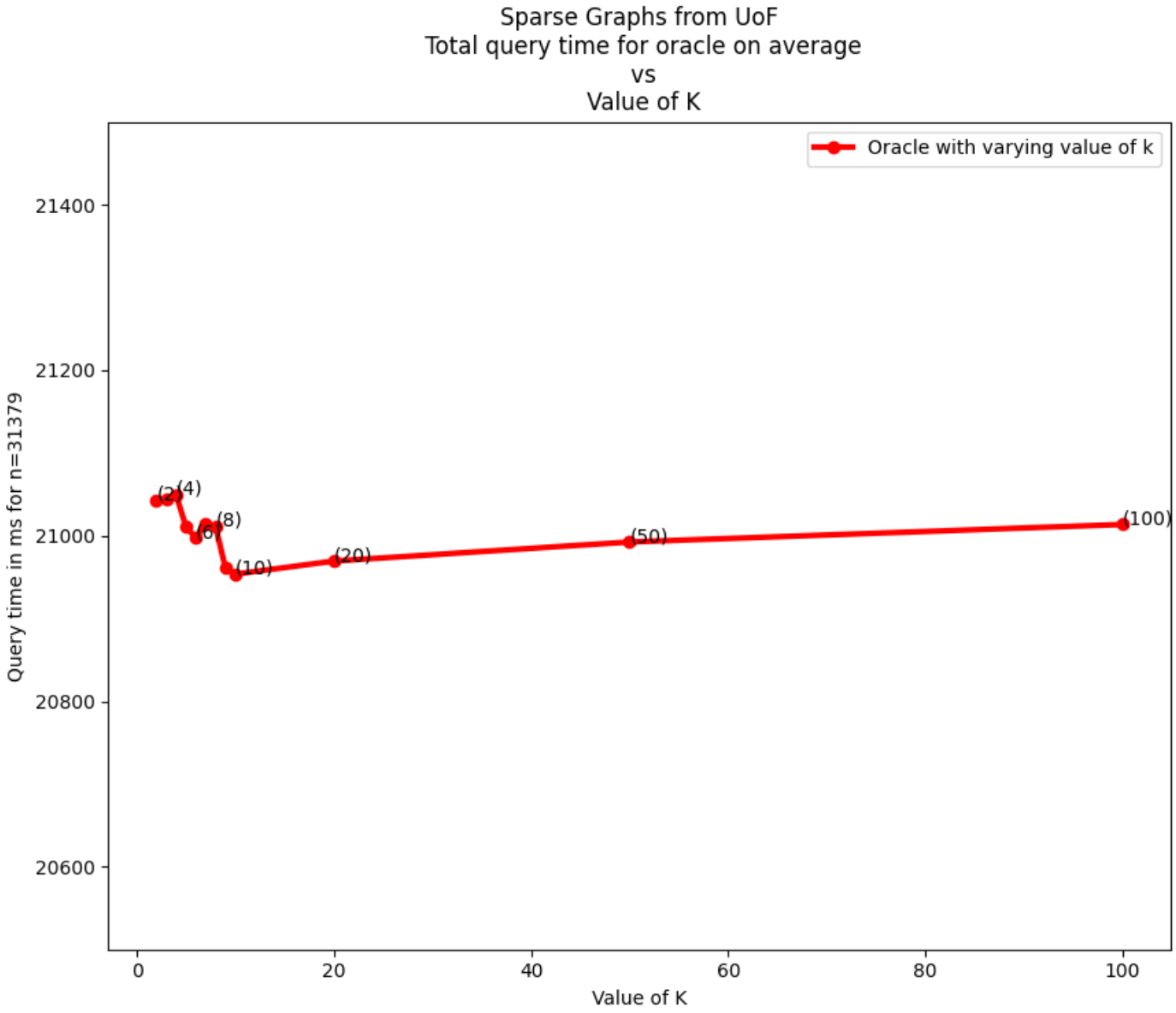
Upper plots represent variation of time of construction of Approximate Distance Oracle, with varying number of edges across different graphs, as well as different values of k. These figures present a curious insight. The complexity of construction, decreases and then again starts to increase, with higher values of k.

Figure-13



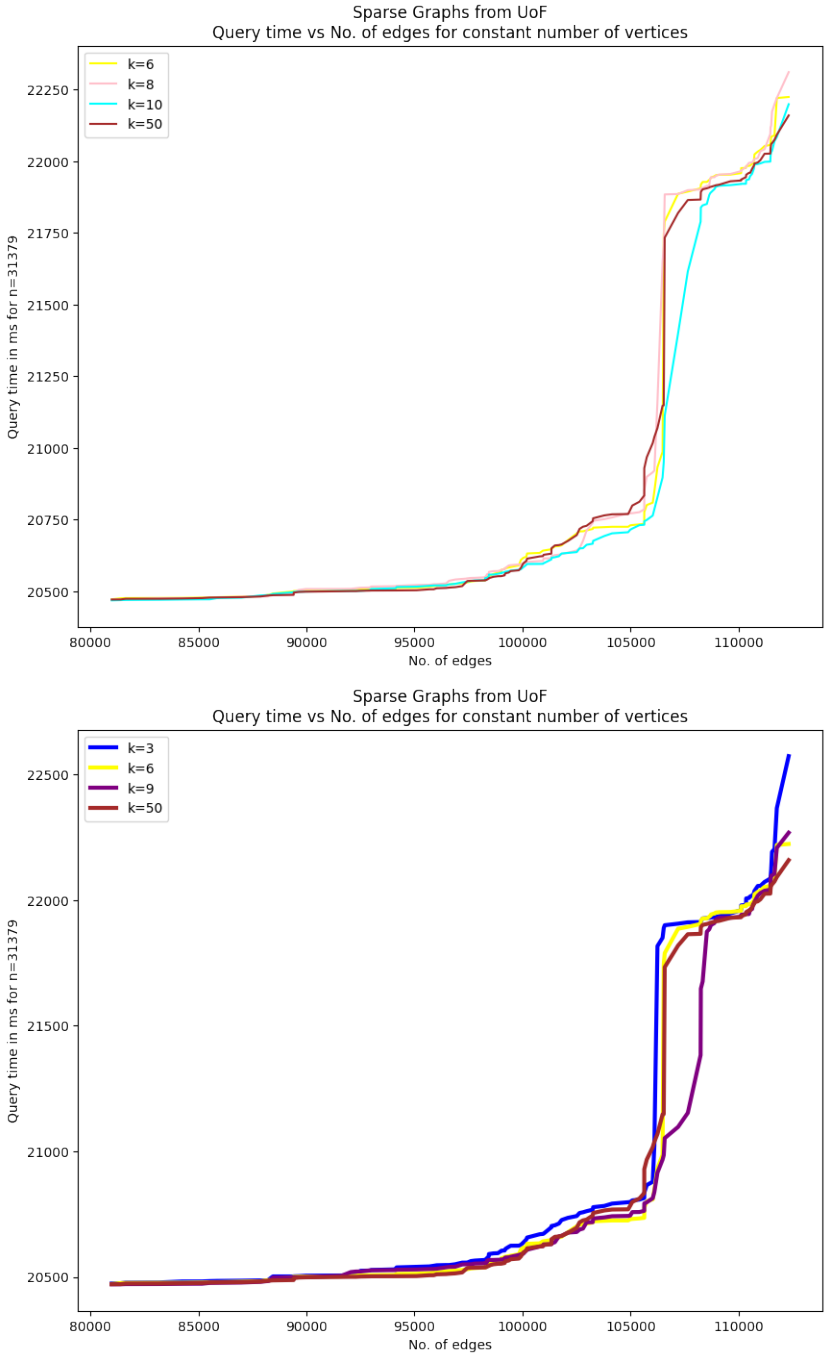
This plot is completely similar to that predicted by $O(kmn^{\frac{1}{k}})$ complexity.

Figure-14



The plot shows that the change is actually minimum and not regular due to some overhead. Some increase may be noticed from 10 to 100, maybe due to the fact that its complexity is $O(k)$

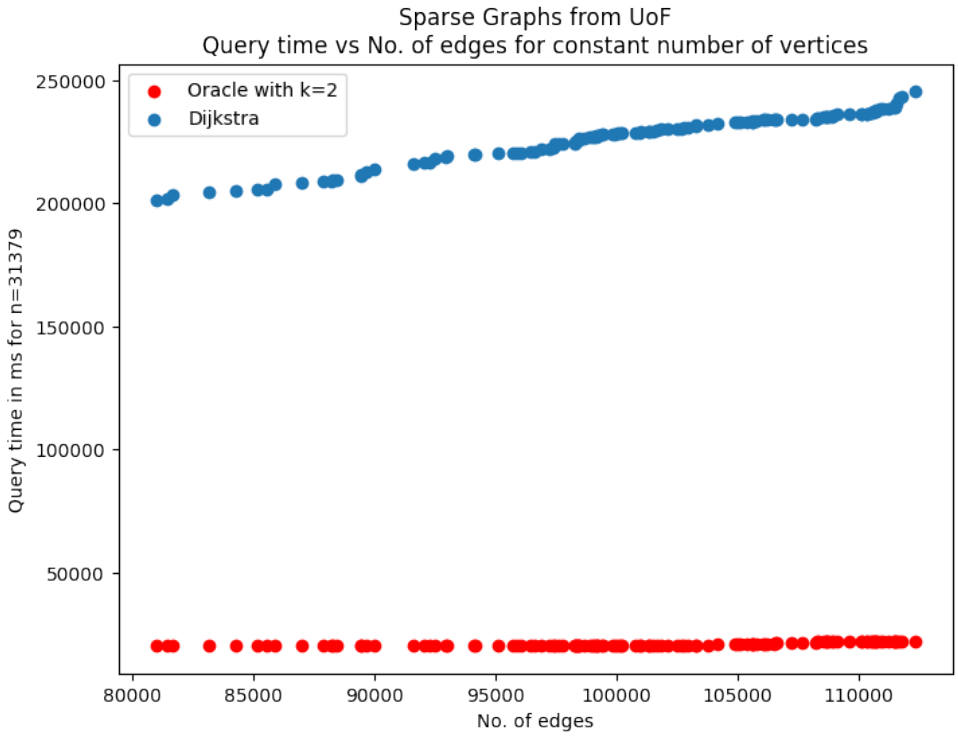
Figure-15 and 16



These plots were showing the dependency of total query time on number of edges. Even if the actual runtime per query stays same, overall time is dependant on $O(n^2)$. The difference across all edges is not clear, but again is minimum for $k = 10$, which is matching with inference coming from Figure-14.

We also ran N-Dijkstras on the same graphs. Output can be summarized in one plot.

Figure-14



In Dijkstra, both query time as well as total query time will be function of number of vertices/edges. For Oracle total query time, will remain constant. It would vary with change in number of vertices. Each query takes constant time, and is not dependant on no. of vertices or edges.

References

[1] Mikkel Thorup and Uri Zwick. "Approximate Distance Oracles". In: *J. ACM* 52.1 (Jan. 2005), pp. 1–24. ISSN: 0004-5411. DOI: 10.1145/1044731.1044732. URL: https://doi.org/10.1145/1044731.1044732.