Efficiently Answering Quality Constrained Shortest Distance Queries in Large Graphs

Zhuo Ma

A thesis in fulfilment of the requirements for the degree of

Master of Philosophy

School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

29/07/2022

ORIGINALITY STATEMENT

☑ I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

COPYRIGHT STATEMENT

AUTHENTICITY STATEMENT

☑ I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis.

UNSW is supportive of candidates publishing their research results during their candidature as detailed in the UNSW Thesis Examination Procedure.

Publications can be used in the candidate's thesis in lieu of a Chapter provided:

- The candidate contributed **greater than 50%** of the content in the publication and are the "primary author", i.e. they were responsible primarily for the planning, execution and preparation of the work for publication.
- The candidate has obtained approval to include the publication in their thesis in lieu of a Chapter from their Supervisor and Postgraduate Coordinator.
- The publication is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in the thesis.

☑ The candidate has declared that **some of the work described in their thesis has been published and has been documented in the relevant Chapters with acknowledgement**.

A short statement on where this work appears in the thesis and how this work is acknowledged within chapter/s:

> The introduction, problem definition, model and experiment of the paper "Efficiently Answering Quality Constrained Shortest Distance Queries in Large Graphs" are contained in Chapter 4,5,6 of my thesis. This paper is submitted to the International Conference on Data Engineering 2022.

Candidate's Declaration

I declare that I have complied with the Thesis Examination Procedure.

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: **Ma**

First name: **Zhuo** Other name/s:

Abbreviation for degree as given in the University calendar: **MPhil**

School: **School of Computer Science and Engineering** Faculty: **Faculty of Engineering**

Title: Efficiently Answering Quality Constrained Shortest Distance Queries in Large Graphs

Abstract

The shortest-path distance is a fundamental concept in graph data analytics and has been extensively studied in literature. In many real-world applications, quality constraints are naturally associated with edges in the graph, and finding the shortest distance between vertices along only valid edges (i.e., edges that satisfy a given quality constraint) is also critical. In this work, we investigate this novel and important problem of quality constraint shortest distance queries. We propose an efficient index structure based on 2-hop labeling approaches. Supported by a path dominance relationship incorporating both quality and length information, we demonstrate the minimal property of the new index. An efficient query processing algorithm is also developed. Extensive experimental studies over real-life datasets demonstrates efficiency and effectiveness of our techniques.

Originality Statement

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

29/07/2022

Copyright Statement

29/07/2022

Authenticity Statement

29/07/2022

Abstract

The shortest-path distance is a fundamental concept in graph data analytics and has been extensively studied in literature. In many real-world applications, quality constraints are naturally associated with edges in the graph, and finding the shortest distance between vertices along only valid edges (i.e., edges that satisfy a given quality constraint) is also critical. In this work, we investigate this novel and important problem of quality constraint shortest distance queries. We propose an efficient index structure based on 2-hop labeling approaches. Supported by a path dominance relationship incorporating both quality and length information, we demonstrate the minimal property of the new index. An efficient query processing algorithm is also developed. Extensive experimental studies over real-life datasets demonstrates efficiency and effectiveness of our techniques.

Publications and Presentations

List of Publications

- Y. Peng, Z. Ma, W. Zhang, X. Lin, Z. Ying, X. Chen, “Efficiently Answering Quality Constrained Shortest Distance Queries in Large Graphs”, submitted to International Conference on Data Engineering(under revision), 2022.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Computing the shortest path distance between two entities in a network is one of the fundamental problems in graph data analytic. Specifically, a path between two vertices s and t is the shortest path if its length is the shortest among all paths between s and t. The distance of the shortest path is called the shortest-path distance, or the shortest distance for short. Due to its optimality, the notion of the shortest distance has been exploited to tackle a broad range of problems, including keyword search [1, 3–5], betweenness centrality [6, 7] and route planning [8].

The shortest distance between two vertices s and t can reflect the vertices' significance. For instance, i) in the nearest keyword search, the vertices closest to the query source are favoured [4], and ii) in social networks, distances are employed in the search ranking to aid users in identifying the most relevant results [1].

Recently, many researchers have studied the efficient processing of shortest distance queries on graph data [9, 10]. However, most of these approaches assume that the network only contains edges without labels. In many real-world applications, many networks' data naturally impose a quality constraint over edges. While computing shortest distances, constraints can be applied on the edge labels to obtain valid shortest distances that are specified for certain applications. For instance, in a road network, road segments may specify the weight limits permitted for auto-trucks. In this scenario, the weight limit is the quality of edges on road networks, and it is demanded to

compute the shortest path by which an auto-truck can pass. Namely, compute the shortest path and the distance along which the auto-truck satisfies the quality constraint of each edge. Another example is to compute the shortest distance on a communication network to determine the most optimal path while restricting all edges on the path to satisfy certain throughput constraints.

To fill in this research gap, we investigated the novel and important problem of quality constraint shortest distance and proposed an efficient 2-hop labeling index approach, namely the WC-Index. Given query vertices, s and t in a graph G and a quality constraint w, the quality constraint shortest distance problem finds the shortest-path distance where the quality of each edge along the path is at least w. Taking advantage of a path dominance relationship incorporating both quality and length information, we developed an efficient indexing construction algorithm for our 2-hop index. Our approach incorporates an extension to the 2-hop index that takes advantage of the ordering of weights and distance, and significantly prunes vertices that would have been processed in a classical 2-hop index while maintaining index minimality. We also investigated the ordering of BFS searches and discovered that using vertex degree or tree decomposition can have different effects on different kinds of networks. An efficient query processing algorithm is also developed. We also demonstrate that our index preserves the property of minimality.

1.1 Applications

Some essential applications are listed as follows:

Communication Networks [11]. To achieve end-to-end Quality-of-Service (QoS) guarantees [12], the transmission of multimedia streams imposes a minimum-bandwidth requirement on all the links of a path. A quality constrained shortest distance query can determine the distance (for the consideration of minimum cost or delay) between two nodes in a network, where each edge/link has a minimum bandwidth demand of w. The resultant path can handle w bits per second for the transmission of a stream, such as audio or video, with guaranteed bandwidth. Figure 1.1 illustrates a motivating example. Given a minimum speed guarantee such as 3 Mbps, a query asks for the distance from R_3 to R_2 with such a speed guarantee. In this case, the resultant distance is 4 since

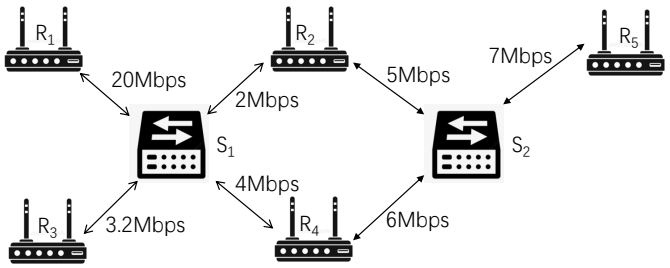


Figure 1.1: A Communication Network. R_i indicates the i^{th} router, and S_i indicates the i^{th} switch.

$R_3 \rightarrow S_1 \rightarrow R_4 \rightarrow S_2 \rightarrow R_2$ fulfills all the criteria, while $R_3 \rightarrow S_1 \rightarrow R_2$ does not owing to the speed of $S_1 \rightarrow R_2 = 2\text{Mbps} < 3\text{Mbps}$.

Social Networks [1]. In social networks, determining the closeness of two individuals is a critical issue. A popular metric is the distance between them in the social networks, e.g., a 2-hop friendship connection is stronger than a 3-hop one. The strength of connections between users is indicated based on profile similarity and interaction activity [13, 14], and the distance between users needs to incorporate such strength information. To support this, a distance query with the quality constraint identifies the distance with only strong connections. Figure 1.2 illustrates an example of friendship network.

Biology Networks [2]. Pathway queries are vital in the analysis of biological networks, where vertices represent the entities, e.g., enzymes and genes, while edges reflect interactions or relations [2]. As shown in [2], one of the four important pathway queries in biological networks is to identify the shortest path between two substances subject to certain constraints. The quality can derive from the activity of kinase [15, 16]. A frequently issued query in these biology networks is to determine the shortest pathway from substance u to transfer to substance v, where all the

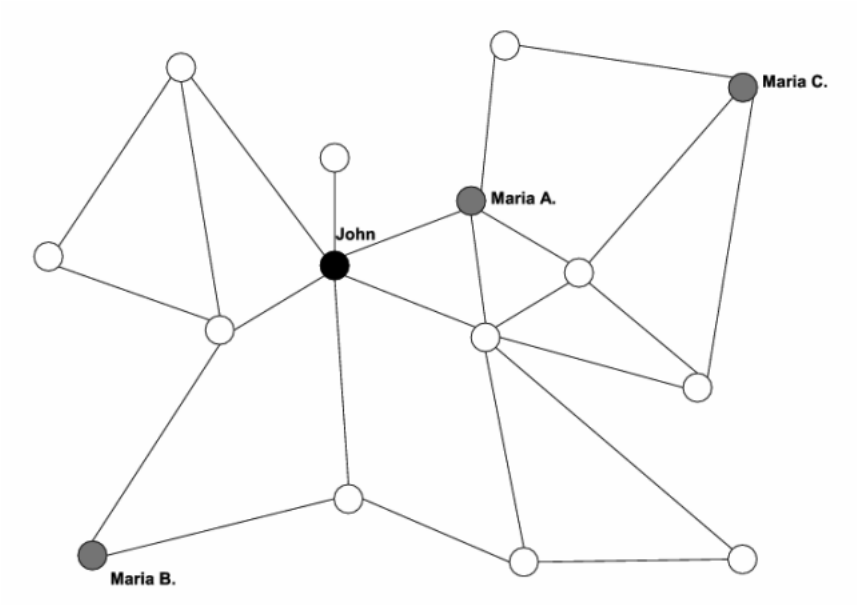


Figure 1.2: An example of social network of "friendship" [1]. In this example, John is at distance 1 of ‘Maria A’, at distance 2 of ‘Maria B’, and at distance 3 of ‘Maria C’.

activities of the kinase on this pathway are as least w.

1.2 Motivations and challenges

To compute the quality constraint shortest distance, a straightforward online approach is to conduct breath first search while taking edge quality into account. Alternatively, one can partition the original graph based on edge quality and perform a breadth-first search on the corresponding graph based on the query. However, for these naïve approaches, the vast search space and the real-time response time are the primary obstacles to this problem. Since there are numerous queries in real applications, the online search algorithms would face the vast search space and could not answer the queries in real time. As for the index-based approach, a naïve index approach involves constructing an index for every possible weight w. In each constructed graph, pairwise distance information is stored. When a query arrives, we could immediately return the results by the corresponding constructed graph’s index. Such a solution could be restricted by the fact that the potential number of w values may be rather large. Consequently, such a naïve index scheme

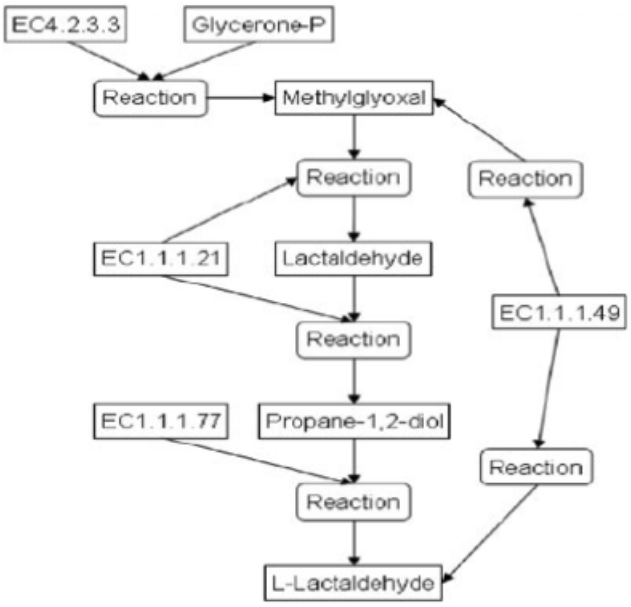


Figure 1.3: Representation of the reactions using the PQL data model [2]. Molecules are boxes, interactions are rounded boxes.

could not satisfy the requirements in the aforementioned applications. We design a modified 2-hop labeling index for the weight-constrained shortest path problem to fill this research gap. To further accelerate indexing time, query time, and to reduce index size, we investigate various pruning rules, propose a query-efficient methodology, and produce efficient vertex ordering.

1.3 Our solutions

To address the above challenges, in this work we adopt the 2-hop labeling index approach to solve the quality constraint shortest distance problem. 2 hop-labeling approaches have been proven to be effective for addressing shortest path problems [17,18]. 2-hop indexes store the distance between vertices that have been precomputed. Each vertex in the index has its own label, which consists of information about distances to other vertices. Queries can be answered by calculations based on the labels of the two query vertices.

The 2-hop index model can be modified by including qualities into index entries, resulting in the labels of vertices containing pairs of distances and qualities as entries. To obtain the con-

strained distance, it performs a similar procedure as for distance queries mentioned previously. The difference is that when a common vertex is found, it checks to see whether both entries have a satisfactory quality value. Our developed WC-Index possesses *soundness*, *completeness*, and *minimal* properties. We investigate the BFS search orders in building the index and propose a quality- and distance-priority-constrained BFS to naturally meet the three properties without incurring additional costs. The query operation over the index is used not only in answering the quality constraint distance queries but also in the index construction phase. We carefully design the query function and achieve linear time complexity by utilizing a nice dominance property of the problem. Last, a hybrid vertex ordering is proposed to tackle both graphs with small and non-small tree width.

1.4 Contributions

Our principal contributions are as follows:

- *Theoretical Analysis*. First, the quality constrained distance problem is defined, which has a variety of applications in road networks, social networks, and biological networks. This paper theoretically analyzes the time and space complexity of this problem. In addition, it investigates the *soundness*, *completeness*, and *minimal* properties, and proposes a sophisticated index capable of naturally preserving these three desirable features.
- *Efficient Index*. We propose a 2-hop labeling based index method. Both query-efficient method and distance-prioritized traversal strategy are presented to expedite index construction. With a nice property of this problem, the query function could be implemented in linear time, which could accelerate both query time and indexing time. Additionally, a hybrid vertex ordering is investigated. In addition, we investigate how to simply modify our index to support the quality constraint shortest path problem.
- *Comprehensive Experiments.* Compared to the baselines, our comprehensive experiments demonstrate the efficiency and effectiveness of our proposed method.

Roadmap. The rest of this thesis is organized as follows. Chapter 2 discusses existing works on the shortest path problem. Chapter 3 introduces the definition of the problem we are trying to tackle. Our 2-hop labeling-based method is proposed in Chapter 4. Chapter 5 presents our experimental results. Chapter 6 concludes the thesis and discusses limitations and future directions.

Chapter 2

Literature Review

In this chapter, we discuss the existing works related to the shortest path problem. We first provide an overview of Index-Free algorithms, which are relatively slow to run. Then, we discuss Index-Based algorithms, which are faster at answering shortest path queries but require pre-computation to build the index. Most shortest path finding algorithms can be categorized into two types. The first type is called the single source shortest path, where the shortest paths from one source vertex to all other vertices are to be calculated. The second type of algorithm is called the all-pairs shortest path, where the paths from every vertex to every other vertex are to be found. Shortest path algorithms can also be classified as exact solutions or approximate solutions. After introducing the fundamental shortest path problem, we present works related to the constraint shortest path problem, which imposes some constraints on the fundamental problem.

2.1 Index-Free Algorithms

First, we introduce the shortest path problem. Given a graph $G(V, E)$ where V is the vertex set and $E = \{(u, v)\}$ where $u, v \in V$ is the edge set. Each edge $e \in E$ is associated with a numeric label that represents its weight. In real-world networks, the weight can represent distance, time, or any other form of cost, where in this case, a larger value weight indicates a higher cost of

traveling along this edge. A path p between the vertex s and the vertex t is a sequence of vertices $\langle v_0, v_1, \cdot, v_k \rangle$ such that $s = v_0$, $t = v_k$ and (v_{i-1}, v_i) is an edge that belongs to $E(G)$ for $\forall i \in [k]$.

2.1.1 Single Source Shortest Path (SSSP)

Definition 1. *(SSSP problem) Given a starting vertex s and a target vertex t where $s, t \in V$, the shortest path algorithm finds the path with minimal combined edge weights between s and t.*

For unweighted graphs, breadth-first search (BFS) [19] is the simplest method to compute the shortest path. By selecting a vertex as starting point s, BFS expands from s by visiting all of its neighbors in each step. Once the target vertex is reached, the minimum distance to the target is the number of steps the expansion has occurred. The path can be deducted by tracing up each vertex's parent. This simple case can be viewed performing calculation on a labeled graph with all edges having a weight equal to one.

In a more practical case, edges contain different numeric weights. Thus, finding the shortest path requires the combined edge weight of the path to be minimal. Dijkstra's algorithm [20] searches the network by the best-first strategy and computes the shortest path from the source vertex to all other vertices. If the destination vertex is given, the algorithm can terminate when the destination is visited. The algorithm starts the shortest path from a source vertex s to all other vertices in the graph. Initially, the distance from s to all vertices is set to infinity, and the source vertex is set as visited. In the first round, it checks and updates all weights from s's neighbors, selects the neighbor with the shortest recorded distance as the next expansion point, and marks it as visited. This process continues until all vertices in the graph are visited. The complexity of Dijkstra's algorithm is $O(n^2)$. One constraint of Dijkstra's algorithm is that it requires all edge weights to be non-negative. A variation of Dijkstra's algorithm for destination-specific SSSP problems searches the network in a bidirectional manner [21].

Figure 2.2 illustrates an example of performing Dijkstra's algorithm to obtain the shortest path and shortest distance from vertex A to all other vertices. The second column of the table stores the flag indicating which vertices have been visited. The third column of the table stores the

shortest total distance from the starting vertex A to other vertices. The fourth column stores the shortest sequence of vertices from A. By backtracking the sequence of previous vertices from the destination, we are able to obtain the shortest path from A to the destination. The detailed steps are as follows.

1. Figure 2.1a consider the starting vertex A. The distance from A to itself is 0, and the distance from A to all other vertices is unknown. Therefore, we set the distance to all other vertices as ∞.

2. Referring to Figure 2.1b. The algorithm then starts by visiting the unvisited vertex that has the smallest distance to the starting vertex A, which is A itself with a distance of 0. Thus, we mark A as visited. From vertex A, we examine all of its unvisited neighbors. We calculate the updated distance between each neighbor of A with the starting vertex (A) by adding the edge weight to the current distance. If the updated distance is less than the known distance, we update the table. The updated distance of B is $0 + 6 = 6$, which is less than ∞. Thus, we update the distance of B as 6. Similarly, we updated the distance of D to $0 + 1 = 1$. We also update the previous vertex of both B and D to A.

3. Referring to Figure 2.1c. The algorithm continues to visit the unvisited vertex with the smallest distance to A, which in this round is D. Thus, we mark D as visited. The unvisited neighbors are B and E. The updated distance to B is $1+2 = 3$, which is less than the current recorded distance of 6. Therefore, we update B's distance to 3. Similarly, we update E's distance to $1 + 1 = 2$. We update the previous vertices accordingly.

4. Referring to Figure 2.2a. In this round, we visit vertex E, which has the smallest distance to A amongst all unvisited vertices. We mark E as visited. The unvisited neighbors are B and C. For vertex B, the updated distance is $2 + 2 = 4$, which is less than the current recorded distance of 3. Thus, we do not update the distance of B. We update the distance of C to $2 + 5 = 7$.

5. Referring to Figure 2.2b. In this round, we visited vertex B and marked it as visited. The only unvisited neighbor is C. The updated distance to C is $3 + 5 = 8$, which is larger than the current recorded distance. Therefore, we do not update it.

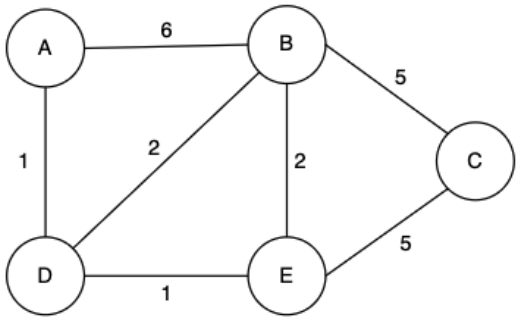
6. Referring to Figure 2.2c. The final unvisited vertex is C. We mark it as visited. Since there are no unvisited neighbors left, the algorithm terminates here.

After the algorithm is terminated, the table of information is complete. The shortest path can be retrieved by backtracking the sequence of previous vertices from the destination. For example, to obtain the shortest path from A to C. We notice that we arrived C via E. This is shown in the previous vertex column of C. When we examine the information for E, we notice we arrived at E via D. Similarly, we arrived D via A. As a result, the shortest path from A to C is $A \rightarrow D \rightarrow E \rightarrow C$.

Fredman et al. developed a new data structure called the Fibonacci heap [22] to improve Dijkstra's algorithm. For a heap with size n, the Fibonacci heap enables arbitrary deletion in $O(logn)$ and other standard heap operations in $O(1)$ time. Therefore, the overall time complexity of Dijkstra's Algorithm improves to $O(nlogn + m)$. Fredman et al., in another work [23] proposed a data structure called AF-Heap, which enables $O(logn/loglogn)$ cost for deletion in a heap and constant cost for other heap operations. The proposed variant of Dijkstra's algorithm has the complexity of $O(m + nlogn/loglogn)$. Based on the idea of the Fibonacci heap, Driscoll et al. [24] introduced the relaxed Fibonacci heap and proposed a parallel variation of Dijkstra's algorithm.

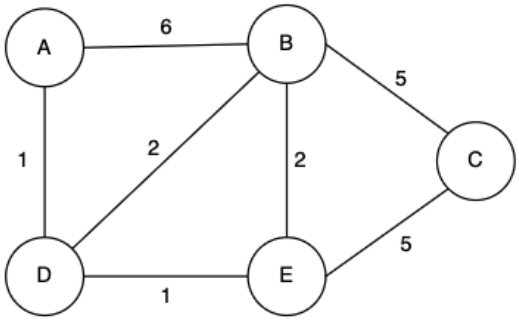
Thorup [25] discovered the relationship between the SSSP problem and the sorting problem and proposed that SSSP should be no harder than sorting edge weights. The paper proposed a priority queue structure that allows a time complexity of $O(mloglogn)$ to compute the SSSP problem. Boas et al. [26] optimize the SSSP problem by improving the implementation priority queue based on a stratified binary tree. The algorithm has a time complexity of $O(loglogn)$ and space complexity of $O(nloglogn)$. Thorup [27] proposed a deterministic linear space algorithm for the undirected SSSP problem. The paper proposed a hierarchical bucket, which is a dynamic set that allows arbitrary insertion and deletion. By utilizing a hierarchical bucketing structure, the algorithm is able to avoid the sorting bottleneck.

One limitation of Dijkstra's algorithm is that it only works on networks with non-negative edge weights. To address this issue, the Bellman-Ford algorithm [28] provides a solution to the SSSP problem that is able to handle negative edge weights. The overall process of the algorithm is



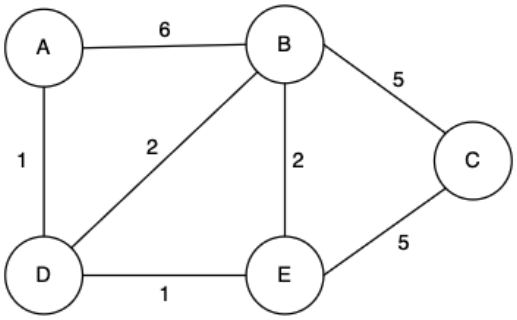
Vertex	Visited	Dist	Prev Vertex
A	F	0	
B	F	∞	
C	F	∞	
D	F	∞	
E	F	∞	

(a) The 1^{st} iteration.



Vertex	Visited	Dist	Prev Vertex
A	T	0	
B	F	6	A
C	F	∞	
D	F	1	A
E	F	∞	

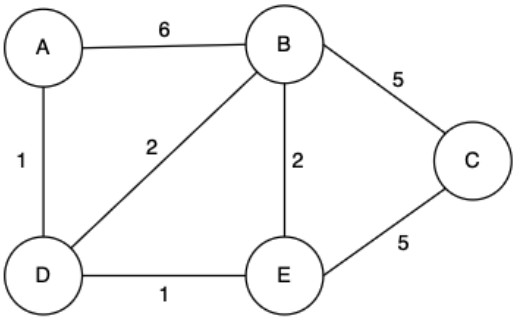
(b) The 2^{nd} iteration.



Vertex	Visited	Dist	Prev Vertex
A	T	0	
B	F	3	D
C	F	∞	
D	T	1	A
E	F	2	D

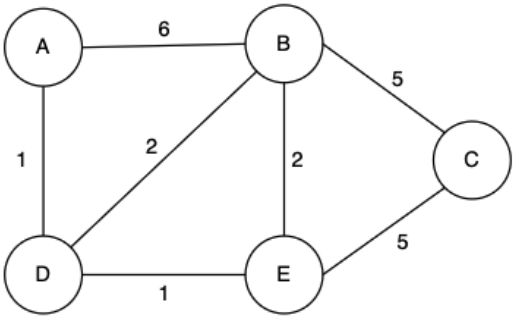
(c) The 3^{rd} iteration.

Figure 2.1: An example of Dijkstra's algorithm.



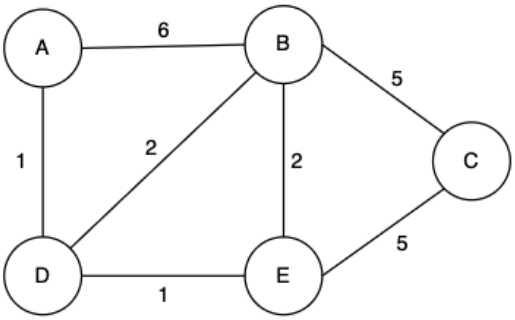
Vertex	Visited	Dist	Prev Vertex
A	T	0	
B	F	3	D
C	F	7	E
D	T	1	A
E	T	2	D

(a) The 4^{th} iteration.



Vertex	Visited	Dist	Prev Vertex
A	T	0	
B	T	3	D
C	F	7	E
D	T	1	A
E	T	2	D

(b) The 5^{th} iteration.



Vertex	Visited	Dist	Prev Vertex
A	T	0	
B	T	3	D
C	F	7	E
D	T	1	A
E	T	2	D

(c) The 6^{th} iteration.

Figure 2.2: An example of Dijkstra's algorithm.

similar to that of Dijkstra's. However, during each step of expansion, instead of selecting the neighbor with the shortest edge distance, the algorithm selects all neighbors, then proceeds in $n-1$ cycles to ensure all changes have been propagated through the graph. The time complexity of the Bellman-Ford algorithm is $O(nm)$. However, if the network contains negative cycles, there are no SSSP solutions due to the lowering of accumulated weights through traversing the cycle. Karp [29] studied the issue of whether the network contains negative cycles. The study proposed the concept of minimum cycle mean, such that finding a negative cycle is analogous to finding a minimum cycle mean.

The A* algorithm is a best-first algorithm [30]. It starts from the source vertex and aims to find the shortest path to the given destination based on heuristics of cost. A tree of paths is maintained beginning at the source vertex and extending all paths edge by edge until termination is reached. In each expansion round, the algorithm decides which edge to expand based on a problem-specific heuristic. More specifically, it selects the path that minimizes a heuristic function $f(n)$, which estimates the total weight of the cheapest path from vertex n to the destination. The algorithm terminates either when the destination is reached, or there is no viable path. A* is guaranteed to return the shortest path from the starting node to the destination if the heuristic function never overestimates the actual cost.

2.1.2 All-Pair Shortest Path (APSP)

Definition 2. *(APSP problem) For a graph $G = (V, E)$, compute all distances between all apris of $s \in V$ and $t \in V$.*

A straightforward approach would be to apply Dijkstra's algorithm separately for every vertex in the graph. Thus, the time complexity of doing so is $O(mn + n^2 log(n))$ [19].

Floyd-Warshall algorithm [31] is a classical solution for finding all pairs' shortest paths in a weighted graph. It can handle graphs with both positive and negative edge weights as long as there are no negative cycles. The algorithm is able to detect negative cycles but is not able to resolve the issue of having them. Although the algorithm only returns the all-pair shortest distance,

simple modification can be performed to reconstruct all the shortest paths. The time complexity of the Floyd-Warshall algorithm is $O(n^3)$ as it compares all possible paths between each pair of vertices. By using a displacement array, the space complexity can be reduced to $O(n^2)$. On sparse graphs where $m << n^2$, running Dijkstra's algorithm on all vertex results in a time complexity of $(mn + n2logn)$, which outperforms the Floyd-Warshall algorithm.

An improved Floyd-Warshall algorithm [9] resolves the issue of the traditional algorithm having too many iterations, which prohibits it from being used on large graphs such as urban road networks. The two significant improvements are: firstly, construct an iterative matrix for solving the shortest path, compare all vertices in the matrix and remove all vertices not in the matrix, then search for the next vertex directly to reduce the number of iterations; secondly, construct a serial number matrix to find the shortest path, it is used to record the case of inserting vertex during iterations.

2.2 Index-Based Algorithms

Index-based shortest-path algorithms pre-compute various kinds of indexes to allow fast query answering. As a general rule, larger indexes take a longer time to be pre-computed but store more information and result in faster query answering.

2.2.1 Separator-Based Algorithms

Separator-Based algorithms divide the network into disjoint partitions and divide the searching process based on the split partitions. Vertices or edges can be regarded as separators that divide the partitions. The shortest paths between vertices of different partitions must pass through their separators. Thus, indexing on the separators is proposed for computing shortest paths.

Van Vliet [32] proposed to utilize vertices as separators that divide graphs into disjoint partitions. The set of separator vertices is regarded as an overlay graph. Pre-computed shortcuts are added between these vertices to preserve the distance of the original graph, thus accelerating pathfinding.

Schulz et al. [33] extend the idea of storing pre-computed shortcuts to multi-level graph structure by allowing shortcuts to be added between levels, where the levels are divided based on a hierarchical decomposition technique proposed by the paper. On the other hand, edge separators are also used as overlay graphs for accelerating pathfinding. Jung et al. [34] divided the graph into sets of edge-disjoint partitions and added shortcuts between the boundaries of each partition. Delling et al. [35] used a similar technique and added real-world optimizations such as turning costs. [36] utilized GPU for further performance improvements.

2.2.2 Hub Labeling Algorithms

Hub Labeling approaches store pre-computed distances as labels for each vertex such that finding the distance between vertices only requires a combination of the labels of each vertex. When answering queries, no computations other than examining the distance entries are needed. It can be viewed as there is a hop vertex between the starting vertex s and destination vertex t.

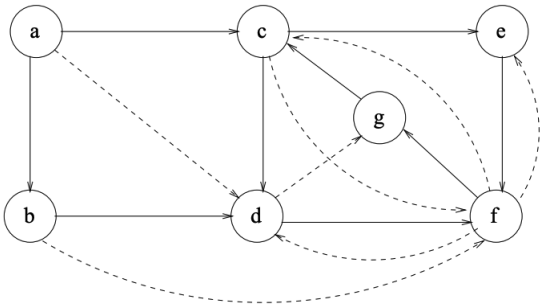
2.2.2.1 Transit Node Routing

Transit Node Routing is one of the first hub labeling techniques proposed [37]. A subset of vertices is regarded as transit nodes while having their all-pair shortest distance computed. While computing the shortest distance between an arbitrary vertex pair $s, t \in V$, if any transit vertex appears on one of the shortest paths, it will be regarded as an access vertex $an(s)$. Then, the shortest distance is simplified to computing $min(s- > a(s)- > a(t)- > t)$. Nevertheless, this algorithm is an approximate algorithm that does not guarantee to return of the correct result.

2.2.2.2 2-hop Labeling

Cohen et al. [38] proposed the 2-hop Labeling and index structure that answers the shortest distance queries correctly with only one intermediate hop vertex. For each vertex $v \in V$, it's label set consists of other vertices and their corresponding shortest distance to v, $L(v) = \{(u, d(v, u))\}$.

For directed graphs, each vertex has an in-label set and an out-label set, each storing the corresponding information based on edge direction. For query $q(s,t)$, the algorithm loops through the labels of s and t to look for common hop vertices w such that $w \in L(s) \cap L(t)$. The shortest distance is obtained by finding the hop vertex that results in the minimum distance.



v	$L_{in}(v)$	$L_{out}(v)$
a		b, c, d
b		f, d
c	f	d, f
d	f, c	g, f
e	c, f	f
f	d, c	g, c, e
g	d, f	c

Figure 2.3: Illustration of a 2-hop index. Edges of the graph are represented by solid lines. Hops that are not edges are represented as dashed arrows.

2.2.2.3 Pruned Landmark

Akiba et al. [18] proposed the Pruned Landmark Labeling for shortest distance computation. It is widely applicable for social networks with low graph diameters as the search space can be pruned dramatically with the help of landmarks. During each iteration of the construction algorithm, a vertex u is chosen as the starting point of Dijkstra's algorithm. The computed shortest distances are added to the in-labels of all other vertices. Then, a backward Dijkstra's is performed, and the resultant distance is added to all other vertices' out-labels. During the searching process, if the distance between u and v can be answered by existing labels, then this process will be pruned. The time complexity of the construction algorithm is $2|V|$ times that of Dijkstra's and becomes faster as the index size grows.

2.2.2.4 IS-Label

For a graph G, and independent set I is a subset of vertices V such that for any $u, v \in I$, $(u, v) \notin E$. Fu et al. [39] proposed the IS-Label, which utilizes the independent set of a graph

for index construction. By extracting independent vertex sets from each level, IS-LABEL organizes the graph into layers that form a hierarchical structure. The remaining vertices at each step are augmented with an edge to preserve distance. Then, labels are constructed with a top-down approach based on the hierarchy. To limit the height of the hierarchy level, IS-Label limits the number of iterations k during the label construction. As a result, instead of building a full index, a residual graph G_k is left in the memory. As the hierarchy is not complete, there may remain a residual graph G_k, where queries performed on it utilizes both the label and a bi-Dijkstra search.

2.2.2.5 Hop Doubling

To address the challenge of large index sizes on large networks, Jiang et al. [40] proposed a hop-doubling method to answer point-to-point path queries for scale-free networks. In contrast to the total label size of $O(|V|^2)$ in the worst case for 2-hop indices, this work derives a complexity bound of $O(h|V|)$ on the index size, where h is a small constant. The algorithm also provides a runtime complexity of $O(|V|logM(|V|/M + log|V|))$, where M is the memory size.

2.2.3 Materialized Approaches

Materialized approaches fully pre-compute the shortest path results and store them to allow nearly constant query performance with a large index size. Sankaranarayanan et al. [41] use path coherence between the shortest path and the spatial positions of vertices on the spatial network to perform compact encoding and fast distance retrieval. Samet et al. [42] proposed an algorithm based on pre-computing the shortest paths between all vertices, taking advantage of the fact that for each vertex u to all other vertices, the shortest paths can be decomposed into subsets based on the first edge from the other vertices. The decomposed paths are organized by a quad-tree, and answering queries involves iteratively visiting neighbors from the corresponding division. Sankaranarayanan et al. [43] proposed Distance Oracle, a framework that computes approximate results by taking advantage of the spatial coherence of the source vertices to reduce space requirements further. Expressly, for source vertices set A and destination vertices set B where they are sufficiently far from each other while the vertices within the sets are close to each other, the shortest paths between any

vertex in A to any vertex in B will be similar. Such an oracle could answer queries in O(logn) time using a B-tree.

2.2.4 Goal-Directed Approaches

Goal-Directed approaches pre-compute index to direct the search space toward the destination. Lauther [44] proposed a modified Dijkstra's algorithm for fast and exact calculation of shortest paths in networks with geometrical information stored at the nodes. By doing so, it is able to prevent redundant costs for expanding in the incorrect directions. The algorithm pre-processes the network by dividing it into regions. Each edge is assigned a label that contains a flag for each region, which indicates whether there is the shortest path into the given region. When calculating the shortest path using Dijkstra's algorithm, only those edges assigned with an appropriate flag need to be investigated. By doing so, the algorithm is able to speed up by a factor of 64 compared to classical Dijkstra implementation. Acceleration is achieved by performing pre-processing on the graph data to create auxiliary information for speed-up shortest path queries.

As Goal-Directed approaches are suitable for solving point-to-point shortest distance problems, researchers have investigated applying this technique, known as the arc-flag approach, to existing shortest path algorithms to enhance performance. Kohler [45] proposed an acceleration method for point-to-point shortest path and constrained shortest path computation in directed graphs. Acceleration is achieved by pre-processing the network to create auxiliary information, which is then used to speed-up shortest path queries. The algorithm focuses on combining multiway-separator with the arc-flag approach. Similar to [44], the arc-flag technique divides the graph into regions and flags edges based on whether it is on the shortest path that leads into specific regions. The multiway-separator also divides the graph along separators and uses that information to improve the shortest path calculation process across different regions. By combining these approaches and bi-directional searches, the algorithm can narrow down the search space for Dijkstra's algorithm. It achieves an average speed-up of up to 1,400 on large road networks.

2.3 Constrained Shortest Path Algorithms

While the shortest path problem focuses on optimizing one goal, which is finding the shortest distance, the Constrained Shortest Path (CSP) problem optimizes based on one objective while requiring other criteria to satisfy some predefined constraints [46]. In such applications, edges $e \in E$ feature multiple labels, in which usually one will be the optimization target (such as cost, distance) while the others are constrained labels. For example, the two objectives can be distance d and some cost c. In this case, we try to find the path that has the shortest distance $d(p)$ while keeping the cost $c(p)$ under particular constraint C. The number of such possible criteria can be unlimited, but in this work, we will focus on one single criterion CSP problem, where there is only one constraint label.

Definition 3. *(CSP problem) For a graph $G(V, E)$ where each edge $e \in E$ is associated with a distance d and cost c, find the path with minimum distance $d(p)$ while keeping the cost $c(p)$ under pre-defined constraint C.*

Figure 2.4 illustrates an example of the CSP problem. Each edge is associated with a label that contains two values. The first value indicates the distance between the two nodes, while the second value indicates the cost of traversing along this edge. Given query $(A, E, 6)$, which indicates finding the shortest path from A to E while keeping the total cost below 6. The answer to this query would be the path $A \rightarrow C \rightarrow E$, as any other paths either have a higher combined weight, or does not meet the cost constraint.

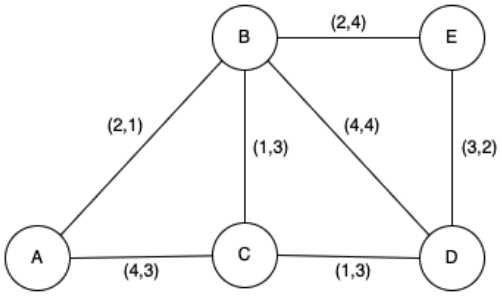


Figure 2.4: An example of the Constrained Shortest Path (CSP) problem.

2.3.1 Exact Algorithms

Hansen [47] proposed the Skyline-Dijkstra to handle multiple criteria pathfinding. Starting from the source vertex, a priority queue is used to store skyline paths. During the search process, each vertex maintains a label set containing all skyline paths passed through this vertex. When a vertex expands its path to its neighbors, it first checks if the proposed path can be dominated by any existing other paths of its neighbor. If not, then this path is added to the label of that neighbor. The search terminates when the priority queue is empty. The complexity of this algorithm is $O(w_{max}|V||E|log(w_{max}|V|)$ where w_{max} is the longest edge distance of the graph.

Kriegel et al. [48] incorporate graph embedding technique to calculate skylines on route selection based on arbitrary network attributes. The core of this skyline query processor is a route iterator that computes top routes according to preference efficiently while avoiding the route computations that need to be issued from scratch in each iteration. In addition, this algorithm is also proposed with pruning techniques to reduce search space. Gong et al. [49] propose a new type of skyline query for finding skyline destinations. The proposed skyline query takes as input a multi-cost transportation network (MCTN), a query point q, and a set of objects of interest D with spatial information. The answer to such queries are objects in D that are not dominated by other objects when considering multiple attributes of these objects and multiple network costs. An exact search algorithm was proposed with the addition of efficient heuristics methods for enhancement.

2.3.2 Approximate Algorithms

Lee [50] proposed an approximate algorithm for multi-constraint decisions named Fallback. Fallback is divided into two phases. In the first phase, it computes the approximate shortest path according to the optimization criteria of the first constraint. If the other constraints are also satisfied, then the result is returned. Otherwise, it re-computes the path based on the second constraint and checks if other constraints are satisfied. This process is repeated until all the constraints are satisfied, or there is no path that satisfies all constraints. As an approximate algorithm, it has no guarantee for the optimal path. Pornavalai et al. [51] modify the algorithm by combining inter-

mediate results. The enhanced algorithm runs a set of forwarding path searches based on each constraint and another set of backward path searches. Then, it finds a relatively optimal path from all combinations of these intermediate results. Ishida et al. [52] proposed a distributed extension of this algorithm. It discusses the issue of constrained least-cost path routing of real-time traffic networks and formulated the problem in a distributed manner. Vertices in the network are distributed across multiple machines. The proposed heuristic algorithm always chooses the shortest path for each intermediate vertex until it satisfies the resource constraint, and is guaranteed to find a delay-constrained path between the source vertex and target vertex.

COLA [53] is a solution for approximate CSP processing on large networks. COLA takes advantage of the fact that there exists a small set of landmark vertices that commonly appear in CSP results. By partitioning the network into regions, CSP is able to index vertices lying on partition boundaries and compute paths within a partition with α-Dijkstra algorithm, which is able to prune paths based on landmarks. Due to the nature of the approximate algorithm, it is able to answer CSP queries in sub-second time.

Chapter 3

Problem Statement and Preliminaries

This chapter introduces the concept of quality constrained shortest distance problem. Part of this chapter comes from the work (Y. Peng, Z. Ma, W. Zhang, X. Lin, Z. Ying, X. Chen, "Efficiently Answering Quality Con-strained Shortest Distance Queries in Large Graphs", submitted to International Conference on Data Engineering(under revision), 2022).

3.0.1 Problem Definition

Quality constrained shortest distance (WCSD) is defined over an undirected unweighted graph $G(V, E, \Delta, \delta)$, where $V(G)$ denotes the set of vertices, $E(G)$ denotes the set of edges, $\Delta \subset \mathbb{R}$ is a set of real-valued qualities, and $\delta : E(G) \rightarrow \Delta$ is a function that assigns each edge $e \in E(G)$ to a real-valued quality $w \in \Delta$. For each vertex $u \in V(G)$, $N_G(u) = \{v|(u, v) \in E(G)\}$ denotes the set of neighbors of u, and $deg_G(u)$ denotes the degree of u, i.e., $deg_G(u) = |N_G(u)|$. A path p from the vertex $s \in V(G)$ to the vertex $t \in V(G)$ is a sequence of vertices $\langle v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rangle$ such that $s = v_0$, $t = v_k$ and (v_{i-1}, v_i) is an edge that belongs to $E(G)$ for $\forall i \in [k]$. The length of p, denoted by $len(p)$, is the number of edges included in the path p, i.e., $len(p) = k$. A path between s and t is the shortest if its length is no larger than any other path between s and t, and the distance between s and t in G, denoted by $dist_G(s, t)$, is defined as the length of the shortest path between s and t.

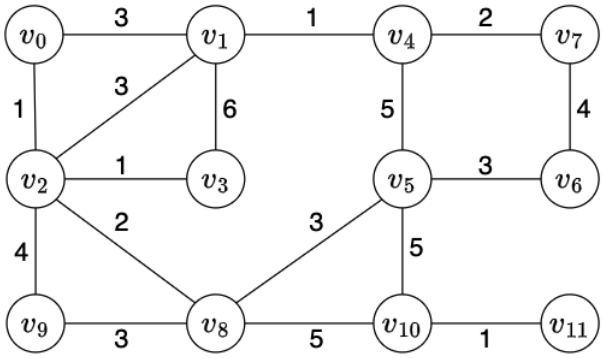


Figure 3.1: An example graph. The values besides edges are their qualities.

Definition 4. *(w-PATH) Given a graph G and a threshold w, a w-path, denoted by p_w, is a path in G such that each of its edge has a quality not smaller than w, i.e., $\forall e \in p_w$, $\delta(e) \geq w$.*

Definition 5. *(w-CONSTRAINED DISTANCE) Given two vertices s and t in a graph G, and a threshold w, the w-constrained distance between s and t, denoted by $dist_G^w(s,t)$, is the minimum length among all the w-paths between s and t.*

For simplify, this work focuses on the distance first. Once the distance is found, the quality constrained shortest path can be easily located.

Definition 6 (WCSD)**.** *Given two vertices s and t in a graph G, and a real-valued threshold w, the WCSD problem is to answer the w-constrained distance query, i.e., computing the w-constrained distance between s and t.*

Example 1. *Figure 3.1 depicts a weighted undirected graph, with the quality of each edge denoted by the number adjacent to it. In this example, a 1-constrained path between v_0 and v_8 is $\{v_0 \rightarrow v_2 \rightarrow v_8\}$ since each edge on the path has a quality no less than 1. It is also the shortest 1-constrained path between v_0 and v_8, therefore $dist_1(v_0, v_8) = 2$. However, $\{v_0 \rightarrow v_2 \rightarrow v_8\}$ is not a 2-constrained path, since the edge (v_0, v_2) has a quality less than 2. Alternatively, $\{v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_8\}$ is the shortest 2-constrained path between v_0 and v_8, therefore $dist_2(v_0, v_8) = 3$. For vertices v_1 and v_4, the path $\{v_1 \rightarrow v_2 \rightarrow v_9 \rightarrow v_8 \rightarrow v_5 \rightarrow v_4\}$ is both a 2-constrained path and a 3-constrained path. However, it is not the shortest 2-constrained path as $\{v_1 \rightarrow v_2 \rightarrow v_8 \rightarrow v_5 \rightarrow v_4\}$ also meets the constraint and has a shorter length.*

3.0.2 2-Hop Labeling Framework

Hub Labeling for Distance Queries. Hub labeling [38] is a vital category of algorithms for distance evaluation. In this class, a label $L(v)$ is computed for each vertex v s.t. the distance between two vertices s and t can be obtained by inspecting $L(s)$ and $L(t)$ only, without traversing the graph. It is NP-hard to generate a labeling with the minimum size [38]. Efficient hub labeling for road networks is explored in [8, 54]. [18] presents a labeling scheme that instead employs paths as hubs. Under the assumption of small treewidth and bounded tree height, [17] proposed a scheme combining both hub labeling and hierarchy for road networks. Pruned landmark labeling (PLL) [55] is the state-of-the-art for real graphs, and its various extensions have been devised. For instance, [40] proposed an external algorithm that generates the same set of labels; [56] devised a parallel algorithm; and [57] describes an algorithm that updates the labels as new edges are inserted into the graph. [58] conducted an experimental study on hub labeling for distance queries.

Chapter 4

WC-Index Solution

This chapter elaborates on our index-based solution to the quality constrained shortest distance problem. Part of this chapter comes from the work (Y. Peng, Z. Ma, W. Zhang, X. Lin, Z. Ying, X. Chen, "Efficiently Answering Quality Con-strained Shortest Distance Queries in Large Graphs", submitted to International Conference on Data Engineering(under revision), 2022).

4.1 Baseline Solutions

4.1.1 Basic Online and Indexing Approaches

BFS-based Online Approaches. A naïve online approach is to conduct a constrained breadth first search, which filters out-edges with quality values lower than the constraint w. The time and space complexity is both $O(|V| + |E|)$. Alternative algorithms such as Dijktra can also be performed. Another solution is to partition the original graph according to the values of quality, then it can perform constrained BFS on the corresponding partition. On large graphs, none of these online approaches is efficient in terms of query time. These algorithms are evaluated as baselines in the experiments.

Details. Algorithm 1 depicts the BFS-based online computation. Line 1 initializes the arrays dis

and $visited$. Line 1 initializes the search queue with the vertex s, and set $visited[s] = true$. Lines 2 to 16 constitute the search procedure. In each iteration, $size$ is set as the current size of P and $dis = dis + 1$ in Line 3. Then, all the vertices are traversed in Line 4 according the to vertex order. For vertex u in P, each vertex v in its neighbors is explored, and it will be pruned in Line 7 if $e(u, v) < w$ or $visited[v]$. Line 10 returns the dis if explores t. Otherwise, w is added into the P and set $visited[v] = true$ to prevent duplication of candidates. INF is returned in Line 17 if not reach t.

Algorithm 1 WC-BFS

Input: any two vertices $s, t \in V$, and constraint w;

Output: $dist^w$ between s and t

1: $dis = 0, \forall v \in V, visited[v] = false, P.push((s)), visited[s] = true$;
2: **while** $P \neq \emptyset$ **do**
3: $size \leftarrow P.size(), dis = dis + 1$;
4: **for** $\forall i \in \{1, .., size\}$ **do**
5: $(u) \leftarrow P.pop()$;
6: **for** $\forall i \in \{1, .., size\}$ **do**
7: **if** $e(u, v) < w$ or $visited[v]$ **then**
8: Continue;
9: **end if**
10: **if** $v == t$ **then**
11: Return dis;
12: **end if**
13: $P.push(v), visited[v] = true$;
14: **end for**
15: **end for**
16: **end while**
17: **return** INF

2-hop Labeling Approach. 2 hop-labeling approaches have been proven to be effective for addressing shortest path problems [17, 18]. 2-hop indexes store the distance between vertices that has been precomputed. Each vertex $u \in V$ has its own label consisting of the form (v, d), where v is another vertex in the graph and d is the distance from u to v. To obtain the distance from vertex s to vertex t, common vertices are identified in the labels of both s and t, calculate the distance as

the sum of their respective distance to w, and return the minimum sum of distances.

Naïve 2-hop labeling method for WCSD. A naïve 2-hop labeling solution involves filtering the graph based on edge qualities and constructing a classical 2-hop labeling index for each filtered graph. $|w|$ is used to denote the number of distinct values of edge qualities. Thus, $|w|$ 2-hop indices will be constructed, each containing only edges satisfying $\forall e \in E, \delta(e) \geq w$. Given a query (s, t, w_0), it can be answered by using the classical 2-hop labeling method by a simple set intersection operation for the corresponding index with w_0. This approach becomes infeasible since the space required to store all of the indices grows as the graph sizes and $|w|$ increase. Moreover, in some instances, e.g., communication networks, the edge qualities are not integers. In such a scenario, it is impossible to create the naïve 2-hop labeling for every possible value of w. The time complexity of the naïve method is $O(|V| \cdot (|V| + |E|) \cdot |w|)$. The space complexity is $O(|V| \cdot |V| \cdot |w|)$ since the number of induced graphs is $|w|$, and in each of them, every vertex could store $|V|$ label entries in the worst case.

4.2 Index Construction

4.2.1 Our proposed 2-hop Labeling Index-based Approach

The naïve approach can answer queries efficiently. Nevertheless, it needs to construct $|w|$ distinct 2-hop indices, which is inefficient and space-consuming. It becomes prohibitive to construct and maintain such a vast number of indices when $|w|$ is large.

Observation 1. *By building $|w|$ 2-hop labeling indexes, one may notice that numerous entries in the separate indices are redundant and elimination of those redundant entries does not impair the correctness of the queries. In this section, a modified 2-hop indexing approach is proposed which seeks to build only one index while efficiently supporting quality constrained shortest distance queries.*

Before providing this approach, the concept of path dominance is firstly described.

Definition 7. *(PATH DOMINANCE) Given two vertices s and t in a graph G, as well as two w-paths from s to t, i.e., p_{w_1} and p_{w_2}, p_{w_1} dominates p_{w_2} if $len(p_{w_1}) \leq len(p_{w_2})$ and $w_1 \geq w_2$.*

Definition 8. *(MINIMAL PATH) A w-path p_w is a minimal path if it cannot be dominated by any other w-path.*

Example 2. *An example is illustrated in Figure 4.1. For paths between vertices v_0 and v_4, path $\{v_0 \rightarrow v_3 \rightarrow v_4\}$ with length 2 dominates path $\{v_0 \rightarrow v_3 \rightarrow v_5 \rightarrow v_4\}$ with length 3, since the two paths have the same minimum edge quality of 1 and the length of $\{v_0 \rightarrow v_3 \rightarrow v_4\}$ is smaller. For paths between vertices v_1 and v_3, $\{v_1 \rightarrow v_2 \rightarrow v_3\}$ with a minimum edge quality of 4 dominates $\{v_1 \rightarrow v_0 \rightarrow v_3\}$ with a minimum edge quality of 1, while both have the same length of 2. Likewise, $\{v_1 \rightarrow v_3\}$ dominates $\{v_1 \rightarrow v_0 \rightarrow v_3\}$ due to both length and minimum edge quality. Path $\{v_0 \rightarrow v_3 \rightarrow v_4\}$ is the minimal 1-path between v_0 and v_4, because it cannot be dominated by any other paths. Also, $\{v_1 \rightarrow v_2 \rightarrow v_3\}$ is both the minimal 3-path and minimal 4-path between v_1 and v_3.*

In this work, the dominance relationship between paths is leveraged and a single compact 2-hop index is generated, which is capable of answering queries regarding arbitrary quality constraint w. The WC-INDEX index is defined as follows:

Definition 9. *(WC-INDEX) Given an undirected weighted graph G, a WC-INDEX $\mathcal{L}$ of G assigns a label set $\mathcal{L}(u)$ to each vertex $u \in V(G)$. An index entry $(v, dist_G^{\bar{w}}(u,v), \bar{w}) \in \mathcal{L}(u)$ indicates that there exists a minimal $\bar{w}$-path between u and v, and it also records the corresponding $\bar{w}$-constrained distance $dist_G^{\bar{w}}(u,v)$ between them.*

Table 4.1: Summary of Datasets

Vertex	$L(\cdot)$
v_0	$(v_0, 0, \infty)$
v_1	$(v_0, 1, 3), (v_1, 0, \infty)$
v_2	$(v_0, 2, 3), (v_1, 1, 5), (v_2, 0, \infty)$
v_3	$(v_0, 1, 1), (v_0, 2, 2), (v_0, 3, 3), (v_1, 1, 2), (v_1, 2, 4), (v_2, 1, 4), (v_3, 0, \infty)$
v_4	$(v_0, 2, 1), (v_0, 3, 2), (v_0, 4, 3), (v_1, 2, 2), (v_1, 3, 4), (v_2, 2, 4), (v_3, 1, 4), (v_4, 0, \infty)$
v_5	$(v_0, 2, 1), (v_0, 3, 2), (v_0, 5, 3), (v_1, 2, 2), (v_1, 4, 3), (v_2, 2, 2), (v_2, 3, 3), (v_3, 1, 2), (v_3, 2, 3), (v_4, 1, 3), (v_5, 0, \infty)$

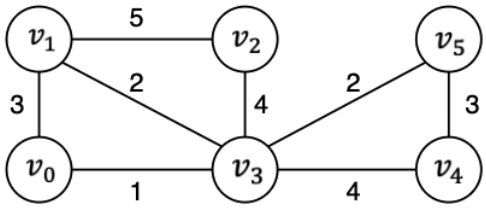


Figure 4.1: A running example.

Given a complete WC-INDEX $\mathcal{L}$ of G, for any two vertices $s, t \in V(G)$ and an arbitrary real-value $w \in \mathbb{R}$, query $Q(s, t, w)$ computes the w-constrained distance between s and t as:

$$dist_G^w(s,t) = \min_{\substack{u \in \mathcal{L}(s) \cap \mathcal{L}(t) \\ w_1, w_2 \geq w}} dist_G^{w_1}(s,u) + dist_G^{w_2}(u,t) \tag{4.1}$$

Algorithm 2 Query Algorithm

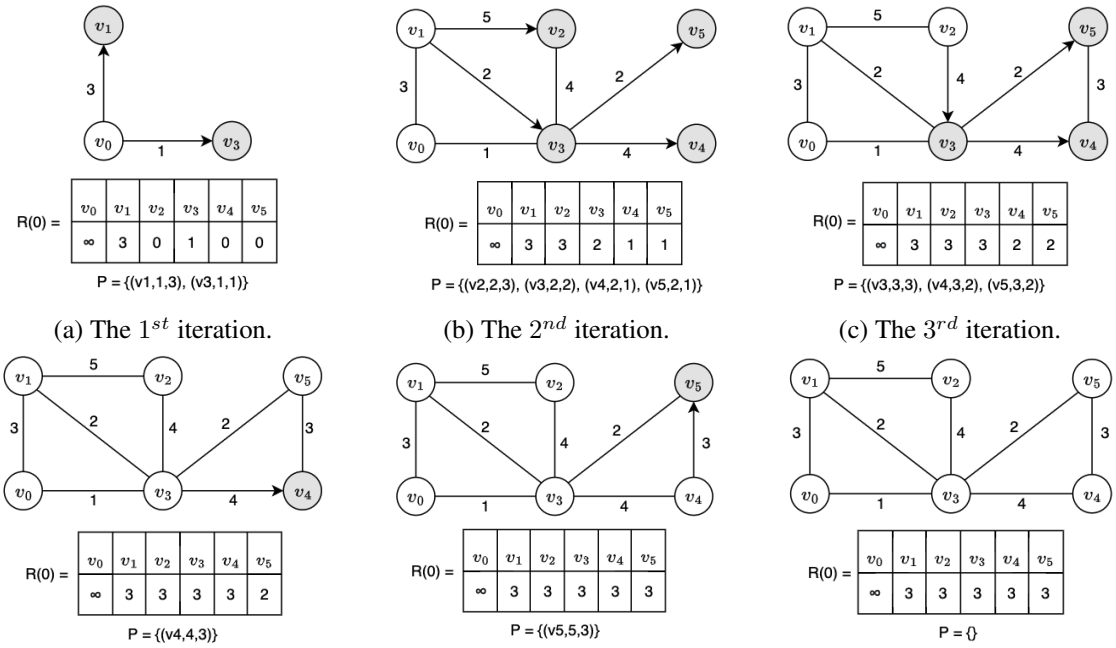
Input:any two vertices $s, t \in V$, and constraint w;

Output:$dist^w$ between s and t

```
1: dist^w ← ∞;
2: for every index entry I_i in L(s) do
3:   if I_i.quality ≥ w then
4:     for every index entry I_j in L(t) such that I_j.vertex = I_i.vertex do
5:       if I_j.quality ≥ w then
6:         if I_i.dist + I_j.dist < dist^w then
7:           dist^w ← I_i.dist + I_j.dist;
8:         end if
9:       end if
10:    end for
11:  end if
12: end for
13: return dist^w;
```

Example 3. *Figure 4.1 illustrates how these 2-hop labeling index works. Given a query $Q(v_2, v_5, 2)$, $L(v_2)$ and $L(v_5)$ are explored. This example starts with the first entry of $L(v_2)$, $(v_0, 2, 3)$, and discover that it satisfies the quality constraint of 2. In the following, entries in $L(v_5)$ are explored that share the same vertex v_0 and also satisfy the quality constraint. $(v_0, 3, 2)$ is the first constraint-*

satisfying entry in $L(v_5)$*. Therefore,* $dist^2 = 2 + 3 = 5$ *is obtained. The next entry* $(v_0, 4, 3)$ *also satisfies the constraint. Nevertheless, since the resultant distance* $dist^2 = 2+4 = 6$ *is larger than the previous distance obtained, no update is performed and* $dist^2$ *remains as* 5*. It then moves on to the second entry of* $L(v_2)$ *which satisfies the constraint:* $(v_1, 1, 5)$*. In* $L(v_5)$*, label entries* $(v_1, 2, 2)$ *and* $(v_1, 4, 3)$ *are found satisfactory and subsequently update the distance as* $dist^3 = 1 + 2 = 3$*. Lastly, it visits* $(v_2, 0, \infty)$ *in* $L(v_2)$ *and finds* $(v_2, 2, 2)$ *in* $L(v_5)$*, resulting in* $dist^2 = 0 + 2 = 2$*.*



(a) The 1^{st} iteration. (b) The 2^{nd} iteration. (c) The 3^{rd} iteration.

(d) The 4^{th} iteration. (e) The 5^{th} iteration. (f) The 6^{th} iteration.

Figure 4.2: The constrained BFS process for v_0.

4.2.2 Distance-Prioritized Search Order

The whole index construction process consists of $|V|$ iterations' constrained BFS starting from different vertices. In each constrained BFS, it will explore at most $|V|$ vertices, but each vertex will be touched at most once. The order of these $|V|$ iterations' starting vertex is named *Vertex Order*, while in i-th constrained BFS starting from vertex v_i, the order to explore remaining vertices is called the *search Order* of v_i. These two types of orders are crucial for indexing time, indexing size, and query time in the 2-hop based index.

Below we introduce three properties that we aim to preserve for WC-INDEX. Then, a smart search order is proposed to guarantee these three properties at no additional cost, particularly the *minimal* property.

- *Soundness*. If there are two index entries $(v, d_1, w_1) \in L(s)$ and $(v, d_2, w_2) \in L(t)$ with $w_1 \leq w_2$ ($w_2 \leq w_1$), then there exists a quality constrained path from s to t with distance $d_1 + d_2$ that satisfies quality constraint $w \leq w_1$ (w_2).
- *Completeness*. If there is a quality constrained shortest path P_0 from s to t with distance d, and satisfying quality constraint w ($w = min(w(e)|e \in P_0)$), then there exist either two label entries $(v, d_1, w_1) \in L(s)$ and $(v, d_2, w_2) \in L(t)$. If $w = min(w_1, w_2)$, then $d_1 + d_2 = d$ and $w = w$, or single label entry such as $(s, d, w) \in L(t)$ or $(t, d, w) \in L(s)$.
- *Minimal*. Intuitively, the minimal property indicates that any deletion of the existing label entries will cause incorrect results for some queries. This property is formulated as follows: For a vertex u, an entry $I = (v, d_1, w_1)$ is minimal if I is not dominated by any other entries in $L(u)$; that is, there is no entry $I' = (v, d_2, w_2) \in L(v)$ s.t. $d_2 \leq d_1$, and $w_2 \geq w_1$. An index entry $I = (v, d_1, w_1)$ is *necessary* if there does not exist a vertex u_0 s.t. $(u_0, d_0, w_0) \in L(s)$ and $(u_0, d_0', w_0') \in L(t)$ where $d_0 + d_0' \leq d_1$ and $min(w_0, w_0') \geq w_0$. Then, a WC-INDEX is *minimal* if every index entry in it is both minimal and necessary.

To efficiently construct the WC-INDEX, the dominance relationships between edge qualities are exploited. Utilizing path domination, pruning is performed by traversing vertices in a certain order. To optimize the number of path traversals that are pruned throughout the index construction process, the following priority-based search orders are strictly adhered:

1. *Distance order*. Computing the index entries with smaller distance d first;
2. *Quality order*. When tackling one specific d value, explore the entries with the largest quality value w first.

Based on the above processing order, the WC-INDEX is constructed using BFS traversals from each vertex. Consider the BFS process from vertex $v \in V$. The maximum w value of paths from

v to all other vertices are recorded. During each iteration of BFS expansion, it will be determined whether the visited vertices, say u, can be reached from v by an existing path that dominates the current path, where an existing path is a path indicated by the current index entries. If the current path from v to u is dominated by an existing path, u is pruned from the BFS process. Otherwise, the corresponding index entries are added into the WC-INDEX. Before moving onto the next iteration, all paths are processed from this iteration of expansion. Therefore, it is guaranteed that the index entries added in this iteration will not be dominated by any other entries.

Algorithm 3 WC-Index Construction

Input:a graph G, and a vertex order $\mathcal{O}$;

Output:the constructed 2-hop index $\mathcal{L}$;

1: $\mathcal{L}(v) \leftarrow \{(v, 0, \infty)\}$ for all $v \in V(G)$;
2: **for** $k = 1, 2, \cdots, n$ **do**
3: $v_k \leftarrow$ the k-th vertex in $\mathcal{O}$;
4: $R(v) \leftarrow 0$ for all $v \in V(G)$;
5: $P \leftarrow$ an empty queues;
6: $P.push((v_k, 0, \infty))$;
7: **while** $P \neq \emptyset$ **do**
8: $vec \leftarrow \emptyset$;
9: **while** $P \neq \emptyset$ **do**
10: $(u, d, w) \leftarrow P.pop()$;
11: **if** QUERY$(v_k, u, w, \mathcal{L}) \leq d$ **then continue**;
12: **else** $\mathcal{L}_k(u) \leftarrow \mathcal{L}_k(u) \bigcup (v_k, d, w)$;
13: **for each** $v_i \in N_G(u)$: $O(v_i) > O(v_k)$ **do**
14: $w' \leftarrow \min(\delta(e = (u, v_i)), w)$;
15: **if** $w' \leq R(v_i)$ **then continue**;
16: $vec \leftarrow vec \bigcup \{v_i\}$; $R(v_i) \leftarrow w'$;
17: **end for**
18: **end while**
19: **for each** $w \in vec$ **do** $P.push(v_i, d + 1, R(v_i))$;
20: **end while**
21: **end for**
22: **return** $\mathcal{L}$;

The algorithm for constructing WC-INDEX is shown in Algorithm 3. Given graph G and a vertex order O, this algorithm constructs the WC-INDEX L, which consists of entry sets $L(v)$ for every $v \in V$. Each entry set $L(v)$ is initialized as a set that contains only one entry, which corresponds to v itself (Line 1). Then, BFS is executed for all $v_k \in V$ following the specified order. A vector $R(v)$ of size $|V|$ is used to record the current largest w value of all paths from v to all other vertices in the graph, with values set to 0 (Line 4). The maximum w value from v_k to u is denoted as w^u_{max}. A queue P is used to store tuples in the form of (u, d, w), where u is a vertex visited in the previous round of BFS, d is the associated BFS path length, and w is the minimum edge quality of that path. P is initialized to contain a single element of $(v_k, 0, \infty)$ (Line 5). The BFS process from v_k is described in Line 7-17 of Algorithm 3.

During each iteration of BFS expansion, for each entry in queue P, a query is performed on the w-constrained path from v_k to u using the current index constructed so far (Line 11). This entry will be pruned if the result w-constrained distance from the query is smaller than the current BFS distance d. If not, the entry is appended to the index (Line 12). Then, for each $v_i \in N(u)$, it will determine whether v_i can be reached from v by an alternative path with a greater w value (Line 13-16). This is determined by comparing the current w to $w^{v_i}_{max}$. If $w < w^{v_i}_{max}$, then w is pruned from the BFS process. Otherwise, v_i is added to a temporary set, and update $w^{v_i}_{max}$ with the value w. After all neighbors of u have been processed, all temporary queue entries are pushed into P to be processed in the next expansion iteration, with a distance of one step further from v_k (Line 17). Thus, the algorithm ensures that for each v_i only one path with the greatest w will be considered in the next iteration. After all potential entries are popped from P, the process is repeated on this queue for the following round of BFS. The entire BFS from v_k ends when P is empty. The construction of WC-INDEX index finishes, after performing BFS for all $v \in V$.

Example 4. *Figure 4.2 illustrates how the Algorithm 3 operates for the vertex v_0 in Figure 4.1. $R(v) \leftarrow 0$ for every $v \in V(G)$ and $R(v_0) = \infty$. Figure 4.2a investigates the neighbors of v_0, i.e., v_1 and v_3. Then, $R(v_1) = 3$ and $R(v_3) = 1$. In addition, P is updated to include the newly added vertices v_1 and v_3. Figure 4.2b indicates that v_2, v_3, v_4, and v_5 will be explored. It is noted that v_3 is updated into P again since in the round, $R(v_3)$ is updated with a larger value, i.e., 2. Likewise, in Figure 4.2c, v_3, v_4 and v_5 are inserted into P due to their updated R values. In Figure 4.2d,*

only v_4 is inserted into P and $R(v_4)$ is updated with 3 since a path $v_0 \to v_1 \to v_2 \to v_3 \to v_4$ is found. In this path, the minimal quality is $e(v_0, v_1) = 3$. Figure 4.2e depicts the updates for v_5 with only v_5 being inserted into P and $R(v_5)$ being updated with value 3. This is the result of the newly found path $v_0 \to v_1 \to v_2 \to v_3 \to v_4 \to v_5$. Figure 4.2f illustrates the last iteration. The constrained BFS for v_0 terminates at this iteration since there is no update for any vertex. For every triple inserted into P, the corresponding label entry is inserted into $L(v_0)$.

Lemma 1. *In Algorithm 3, for each candidate index entry popped by the queue (Line 10), it cannot be dominated by all the candidate index entries popped by the queues afterwards.*

Proof. Since for each iteration of BFS expansion, all entries in queue P are popped, and then new entries are added back to this queue P. Then, this queue can only contain entries with the same d at a given moment. For vertex u in a certain BFS iteration, if u already exists in the temporary set, which indicates there is an existing path from v_k to u with w^u_{max}. If the current path induces an entry with $w > w^u_{max}$, w^u_{max} is updated to be w. Otherwise, nothing happens. Consequently, w will only exist once in the temporary set, and will only be pushed into queue P once, with $w = w^u_{max}$, which is the maximum w value at distance d. Any future entries in P regarding u will include a larger d. Therefore, popped entries will never be dominated by future entries in the queues. □

Correctness of Algorithm. Then, the correctness of the algorithm is proved by its *Soundness* and *Completeness*. Additionally, the *Minimal* properties is proved.

Theorem 1. *Algorithm 3 can construct a Sound, Complete, and Minimal index for WCSD problem.*

Proof. First, the *Soundness* and *Completeness* are demonstrated. These two characteristics are equivalent to the correctness of Algorithm 3.

Soundness. It is proved by contradiction. Assume there are two index entries $(v, d_1, w_1) \in L(s)$ and $(v, d_2, w_2) \in L(t)$ with $w_1 \leq w_2$ ($w_2 \leq w_1$), and there does not exist a quality constrained path from s to t with distance $d_1 + d_2$ and satisfy quality constraint $w \leq w_1$ (w_2). According

to the index construction process, there are two quality constrained shortest paths. The first is $s \rightsquigarrow v$ with distance d_1 and quality constraint w_1, whereas the second is $v \rightsquigarrow t$ with distance d_2 and quality constraint w_1. Therefore, it can be combined to produce a new path P_{new}. Note that *soundness* simply requires a quality constraint path; it does not have to be shortest.

Completeness. Similarly, the *Completeness* is demonstrated by contraction. Assume that there is a quality constrained shortest path from s to t with distance d, satisfying quality constraint w, then there does not exist either two label entries $(v, d_1, w_1) \in L(s)$ or $(v, d_2, w_2) \in L(t)$. If $w = min(w_1, w_2)$, then $d_1 + d_2 = d$, nor one label entry like $(s, d, w) \in L(t)$ or $(t, d, w) \in L(s)$. Assume s is explored before t[1] and s is the first vertex that leads to such incorrectness, and s, t is the first vertex pair to lead the incorrectness. This indicates that the *Completeness* of all the previously explored vertices is maintained. Consequently, according to Algorithm 3 Line 11, if the $Query(s, t, w)$ is pruned, then it indicates that there exist two label entries $(v, d_1, w_1) \in L(s)$ and $(v, d_2, w_2) \in L(t)$. If $w = min(w_1, w_2)$, then $d_1 + d_2 = d$. Otherwise, the label entry (s, d, w) is inserted into $L(t)$ in accordance with Algorithm 3 Line 12.

Minimal. According to Algorithm 3 Line 11, a newly added label entry is *Minimal* when it is inserted into the index. Therefore, it is only necessary to prove it will not be dominated in the label entries that are inserted after it. Due to the *distance order*, *quality order*, and Definition 7, this property is automatically maintained. □

Theorem 2. *The index constructed by Algorithm 3 is capable of producing correct results.*

Proof. Theorem 1 proves the *Soundness* and *Completeness* of constructed index. Thus, its correctness is immediately proved. □

Complexity Analysis. The while loop dominates the time complexity of indexing from the vertex v_k. Let $I(v)$ denote all the index entries associated with v and let $\zeta = \max_{v \in V(G)} |I(v)|$. Let d_{max} denote the maximum vertex degree in the graph. Observe that in Algorithm 3, Lines 13-16 are executed at most ζ times, hence the size of the priority queue cannot exceed ζd_{max}. For each index

[1]The proof process is similar if t is the earlier one.

entry in the queue, a query operation is performed to determine whether it can be covered by the existing index entries, and the query time is bounded by $O(\zeta)$. As a result, the time complexity of Algorithm 3 is $O(n \cdot \zeta \cdot d_{max}(\log \zeta \cdot d_{max} + \zeta))$.

The size of the index is bounded by $O(\sum_{u \in V(G)} \sum_{v \in V(G)_{\leq u}} min(D, |w|))$.

4.2.3 Query-Efficient Implementation

Since the $Query$ function is commonly utilized during the index construction and query stages, it is a vital component that influences three aspects of the index: indexing time, index size, and query time. This subsection investigates how to efficiently implement the $Query$ function by utilizing the problem's property.

Given a query(s, t, d, w), a basic operation is to determine whether there are two label entries (u_1, d_1, w_1) and (u_2, d_2, w_2) with $u = u_1 = u_2$, $d_1 + d_2 \leq d$, $w_1 \geq w$, and $w_2 \geq w$.

Naïve Implementation. For simplicity, $L[u]$ denotes all the label entries of vertex u, and $L[u][v]$ denotes all the label entries as (v, d_v, w_v) in $L[u]$. The naïve query function is represented by Algorithm 4. Line 1 traverses every label entry in the $L[t]$. Assume a label entry is I_j, Line 3 prunes it if its vertex order is larger than s or $quality$ is less than the quality constraint. Otherwise, entries of $L[s][v]$ are explored, where v is the vertex of I_j, and validate whether there are two valid label entries I_j and I_i to return a $true$ result. The time complexity of this implementation is $O(|L(s)| + |L(t)| + \sum_{v \in L[t].vertex} |L[t][v]| \times |L[s][v]|)$.

The following theorem helps speed up this procedure.

Theorem 3. *For two label entries (u_0, d_0, w_0) and (u_0, d_1, w_1) in $L(v)$, if $d_0 > d_1$, then $w_0 > w_1$, and vice versa.*

Proof. This theorem is proved by contradiction. Assume that there are two label entries (u_0, d_0, w_0), and (u_1, d_1, w_1), s.t. $d_0 > d_1$ and $w_0 \leq w_1$. According to Lemma 1, (u_0, d_0, w_0) will be eliminated since it is dominated by (u_1, d_1, w_1), which results in a contradiction. Likewise, a similar contradiction exists when $w_0 > w_1$ with $d_0 \leq d_1$. □

Algorithm 4 $Query$

Input:any two vertices $s, t \in V$, constraint w, and current distance d;

Output:a boolean value indicating if a path is found;

```
1: for ∀I_j ∈ L[t] do
2:     if I_j.vertex > s or I_j.quality < w then
3:         continue;
4:     end ifv = I_j.vertex;
5:     if L[s][v] = ∅ then
6:         continue;
7:     end if
8:     for ∀I_i ∈ L[s][v] do
9:         if I_i.quality ≥ w then
10:            if I_i.dist + I_j.dist <= d then
11:                return True;
12:            end if
13:        end if
14:    end for
15: end for
16: return False;
```

Querying. During the BFS process for one vertex v_k, it is noted that all queries are issued with one end-point as v_k. Therefore, an array T of size $|V|$ is initialized with the existing index entry of that v_k before the BFS begins. To evaluate $QUERY(u, v_k)$, the new querying algorithm needs $O(|L(u)|)$ time rather than $O(|L(u)|) + O(|L(v_k)|)$ for looping through two entry lists.

Based on Theorem 3, the index entries $(d_i, w_i), i = 1, 2, ..., |L(v)|$ for vertex $v \in V$ must be in increasing order in terms of d and w. If $j > i$, then both $d_j > d_i$ and $w_j > w_i$. Instead of iterating through T, binary search could be utilized to locate elements. Then, in such an implication, the time complexity is $O(|L(s)| + |L(t)| + \sum_{v \in L[t].vertex} |L[t][v]| \times log|L[s][v]|)$.

Query-Efficient Implementation. Based on Theorem 3, the time complexity can be further reduced to $O(|L(s)|+|L(t)|)$. The idea is explained as follows: Since the index entries $(d_i, w_i), i = 1, 2, ..., |L(v)|$ for vertex $v \in V$ must be in ascending order in terms of d and w, if finding the first index entry (u_i, w_i, d_i) whose $w_i \geq w$, d_i is minimal for $(u_i, \cdot, \cdot)$. Thus, for every $v \in L(s)$ or

$L(t)$, only one label entry is required. Then, a naïve scanning could be conducted to answer the queries. The time complexity is $O(|L(s)| + |L(t)| + \sum_{v \in L[t].vertex} (log|L[t][v]| + log|L[s][v]|))$. Since $\sum_{v \in L[t].vertex} (log|L[t][v]| + log|L[s][v]|) \leq |L(s)| + |L(t)|$, the final time complexity is $O(|L(s)| + |L(t)|)$.

Details. The details of the Query-Efficient Implementation is illustrated in Algorithm 5. Line 1 traverses every vertex $v \in L(t)$. Line 3 prunes if $L[s][v] = \emptyset$. If not empty, a modified binary search is utilized to locate the first label entry with $w_i \geq w$ in $L[t][v]$ and $w_j \geq w$ in $L[s][v]$, respectively. $true$ is immediately returned if $d_1 + d_2 \leq d$ in Line 7. Otherwise, the query answer is $false$ in Line 8;

Algorithm 5 $Query^+$

Input:any two vertices $s, t \in V$, constraint w, and current distance d;

Output:a boolean value indicating if a path is found;

```
1:  for ∀ vertex v ∈ L[t] do
2:      if L[s][v] = ∅ then
3:          continue;
4:      end if
5:      Find I_i ∈ L[t][v] which is the first label entry with w_i ≥ w;
6:      Find I_j ∈ L[s][v] which is the first label entry with w_j ≥ w;
7:      if d_i + d_j ≤ d then
8:          return True;
9:      end if
10: end for
11: return False;
```

Efficient Initialization. An important aspect is to avoid $O(n)$ time initialization for data structures during each round of BFS. This may develop into a bottleneck. A solution is to set updated values in the array, without recreating the whole array. This can be accomplished by recording which vertices have been processed during the process, and only update them.

Further Pruning. Whenever a path is found during the query process of index construction, the algorithm records the result for the current quality of that vertex pair. If a potential query in the same BFS round has the same vertex pair and a quality not greater than the recorded quality, the

query process can be skipped since its result is recorded.

4.2.4 Vertex Ordering Strategies

Vertex ordering is one of the vital orders that significantly affect indexing time, index size, and querying time. This subsection investigates a hybrid vertex ordering based on some observations.

Observation 2. *The degree ordering is shown to have better performance than other orderings [18] for the shortest path distance problem in the scale-free network, e.g., social networks. Notwithstanding, for the road network "Indochina"[2], the tree decomposition based ordering has much better performance.*

Observation 3. *It is shown in [17] that Vertex Hierarchy via Tree Decomposition technique is appropriate for the road network for distance query.*

To use the Observation 3, it first introduces the degree-based ordering as well as the Vertex Hierarchy through Tree Decomposition.

Degree-Based Scheme. A vertex with a higher degree is likely to cover more shortest paths. In summary, in degree-based ordering, vertices are sorted in non-ascending order of degree. This scheme leads to the state-of-the-art canonical hub labeling for shortest distance queries.

Tree Decomposition Ordering. Tree decomposition is a technique for mapping a graph to a tree in order to accelerate the resolution of certain computational problems in graphs [59,60]. Numerous algorithmic problems, such as maximum independent set and Hamiltonian circuits that are NP-complete for arbitrary graphs, can be solved efficiently by dynamic programming for graphs of finite treewidth, employing the tree-decompositions of these graphs. A summary of Bodlaender's introduction can be found in [61]. The tree decomposition provides a natural hierarchy to vertices. In this work, tree decomposition is utilized to establish the vertex hierarchy, and demonstrate that the hierarchy is effective in resolving quality constrained distance queries in networks. A tree decomposition of a graph $G(V, E)$ is defined as follows [61]:

[2] http://law.di.unimi.it

Definition 10 (Tree Decomposition)**.** *A tree decomposition of a graph $G(V, E)$, denoted by T_G, is a rooted tree in which each node $X \in V(T_G)$ is a subset of $V(G)$ (i.e., $X \subset V(G)$) with the following three conditions:*

- $\bigcup_{X \in V(T_G)} X = V$;
- *For every $(u, v) \in E(G)$, there exists $X \in V(T_G)$ s.t. $u \in X$ and $v \in X$.*
- *For every $v \in V(G)$ the set $\{X | v \in X\}$ forms a connected subtree of T_G.*

Based on Observations 2 and 3, it simply employed vertex ordering of the Vertex Hierarchy via Tree Decomposition in [17] and developed a fast approach to obtain this ordering as opposed to constructing their whole index for the WCSD problem.

The computation of the treewidth of a graph has been shown to be NP-Complete [62]. One of the most effective heuristics Tree decomposition is based on minimum degree elimination.

Minimum Degree Elimination (MDE)-based Tree Decomposition. Minimum Degree Elimination [63] based tree decomposition removes recursively the vertex v in G with the minimum degree and then adds v's neighbors' clique back to G. A bag of the tree decomposition is comprised of each node v and its neighbors on the transient graph right before the deletion of v.

Definition 11 (Minimum Degree Elimination)**.** *Generate n bags of nodes $\{B_1, B_2, ..., B_n\}$ and a sequence of nodes $\{v_1, v_2, ..., v_n\}$ in n rounds with the starting graph $G_0 = G$. In the $i - th$ round, i takes value from 1 to n:*

- *v_i : the node with the lowest degree (or any one of these nodes if there is a tie situation) in G_{i-1}.*
- *N_i : the neighbor set of v_i in G_{i-1}.*
- *B_i : $\{v_i\} \cup N_i$.*
- *G_i : a graph that eliminates v_i from G_{i-1} and then adds clique(N_i), that is $V(G_i) = V(G_{i-1} \backslash \{v_i\}$, and $E(G_i) = E(G_{i-1}) \cup E[clique(N_i)] \backslash \{N_i\} \times N_i$.*

Hybrid Vertex Ordering. Therefore, this work proposes a hybrid vertex ordering that compromises between the computational efficiency of degree vertex order and the index size effectiveness of the tree decomposition order as follows:

- *Classification*. All vertices are classified into two categories: core part and periphery. To achieve this, a degree threshold δ is specified. If a vertex v's degree is above this threshold, it is classified into the core-part. Otherwise, it is classified into the periphery.
- *Core-Part*. Regarding the core-part vertices, it is observed that the computation cost can be quite high if the tree decomposition method is used. Therefore, all these vertices are ordered according to their degree.
- *Periphery*. The vertices in periphery are ranked according to tree decomposition order.
- *Combinations*. Then, these two types of vertices are combined to produce a hybrid vertex order.

Chapter 5

Experimental evaluations

This chapter evaluates the WC-Index through extensive experiment. Part of this chapter comes from the work (Y. Peng, Z. Ma, W. Zhang, X. Lin, Z. Ying, X. Chen, "Efficiently Answering Quality Con-strained Shortest Distance Queries in Large Graphs", submitted to International Conference on Data Engineering(under revision), 2022).

5.1 Datasets

Datasets. Tables 5.1 and 5.2 provide the statistics of real graphs used in the experiments. 14 publicly available datasets are used. These datasets can be downloaded from either KONECT [64][1] or SNAP [65][2]. Directed graphs were converted to undirected ones in our testings. For query performance evaluation, 10,000 random queries were employed and the average time is reported. For labelled graphs such as Movielens, |w| is directly taken from the original dataset. For other non-labelled graphs, we randomly generate those weights.

[1] http://konect.uni-koblenz.de

[2] https://snap.stanford.edu

5.2 Experimental Settings

Settings. In experiments, all programs were implemented in standard c++11 and compiled with g++4.8.5. All experiments were performed on a machine with 20X Intel Xeon 2.3GHz and 385GB main memory running Linux(Red Hat Linux 7.3 64 bit).

Table 5.1: Summary of Road Networks

Name	**Dataset**	$\|E(G)\|$	$\|V(G)\|$
NY	New York City	264,346	733,846
FLA	Florida	1,070,376	2,712,798
CAL	California and Nevada	1,890,815	4,657,742
E	Eastern USA	3,598,623	8,778,114
W	Western USA	6,262,104	15,248,146
CTR	Central USA	14,081,816	34,292,496
USA	Full USA	23,947,347	58,333,344

Table 5.2: Summary of Social Networks

Name	**Dataset**	$\|E(G)\|$	$\|V(G)\|$	$\|w\|$
MV-10	Movielens-10m	80,555	10,000,054	5
EU	eu-2005	862,664	16,138,468	3
ES	eswiki-2013	970,331	21,184,931	3
MV-25	Movielens-25m	221,588	25,000,095	5
FR	frwiki	1,350,986	31,037,302	3
UK	uk-2007	1,000,000	37,061,970	3
SO-Y	Stackoverflow (year)	28,183,518	2,601,977	9

5.3 Baseline Algorithms

Algorithms We compare our techniques with the following baseline solutions.

- **W-BFS**. The original graph is partitioned into $|w|$ parts, and then conduct BFS.
- **Dijkstra**. After the partitioning of the original graph into $|w|$ parts, Dijkstra is conducted.
- **C-BFS**. It conducts Constrained BFS on the original graph, with the valid edges explored.

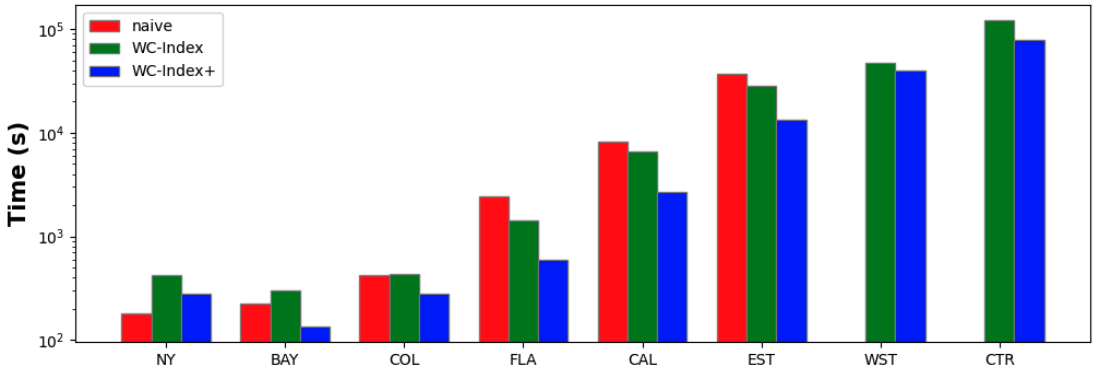


Figure 5.1: Indexing Time (s) for baseline, WC-INDEX, and WC-INDEX+.

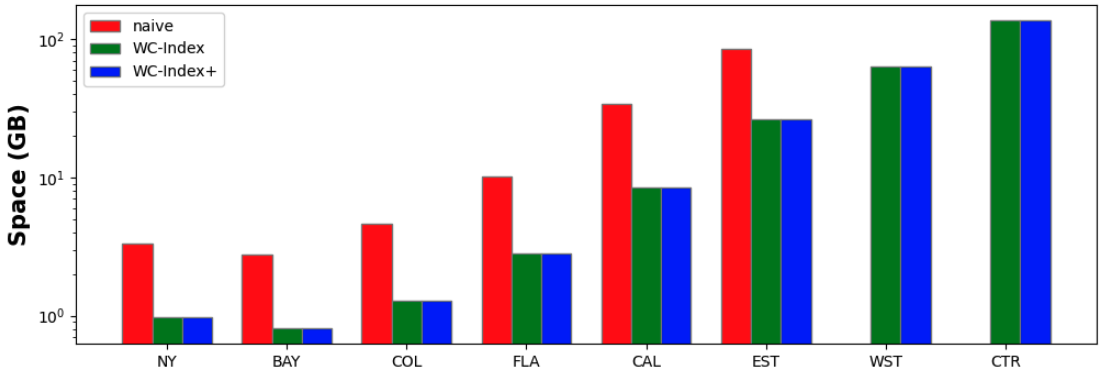


Figure 5.2: Indexing Size (GB) for baseline, WC-INDEX, and WC-INDEX+.

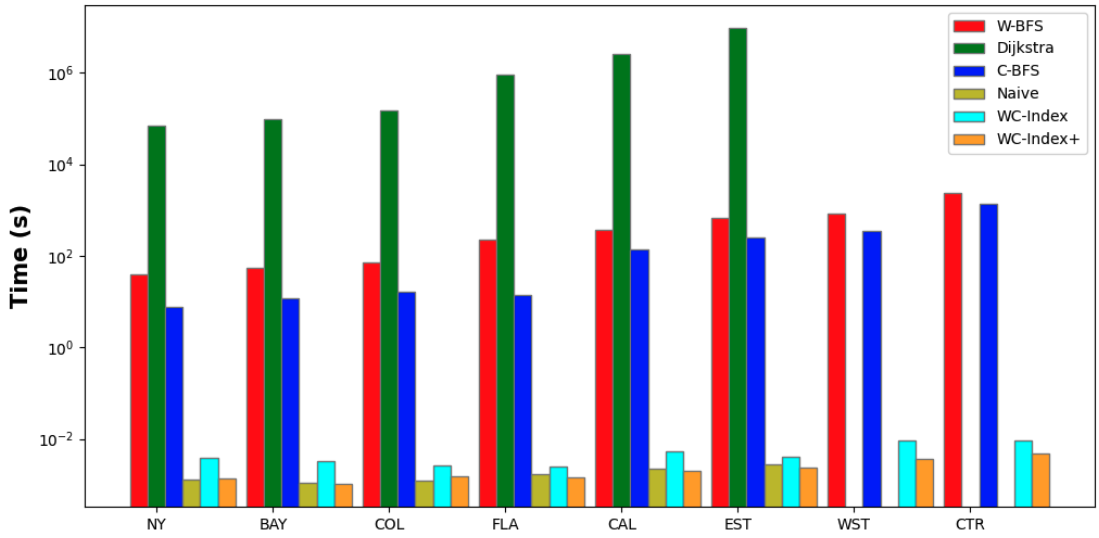


Figure 5.3: Querying time (ms) for baselines, WC-INDEX, and WC-INDEX+.

- **Naïve**. The naïve 2-hop labeling method introduced in Section 4.1.
- **WC-INDEX**. The basic algorithm for the quality-constrained shortest path problem. e
- **WC-INDEX+**. The advanced algorithm with the query-efficient and hybrid order techniques.

5.4 Evaluation on Road Network

5.4.1 Index Construction

Exp 1: Indexing Time for Road Networks. Figure 5.1 illustrates the indexing time for Naïve 2-hop labeling index, WC-INDEX, and WC-INDEX+. What stands out in these figures is that WC-INDEX+ is the fastest method to construct the index among these three algorithms. For instance, for CTR, only WC-INDEX+ can construct the 2-hop index. As for Naïve and WC-INDEX, WC-INDEX is slower than Naïve in small datasets, e.g., NY, BAY, COL, EST. Notwithstanding, WC-INDEX is much faster than Naïve for large datasets, e.g., WST and CTR. We observed that for smaller graphs, the construction overhead of WC-INDEX dominates the index construction time. As a result, WC-INDEX builds up slower than the baseline index. On the other hand, na ¨ıve index simply filter the graph based on every possible weight and construct simple 2-hop indexes for each filter graph. When the graphs are small, this can be done relatively quickly compared to WC-INDEX. However, as the size of the graphs gets large, building indexes for every separate filtered sub-graph is costing much more time, and is eventually outperformed by WC-INDEX, b which only constructs one index.

Exp 2: Indexing size for Road Networks. Figure 5.2 depicts the index size for Naïve 2-hop labeling index, WC-INDEX, and WC-INDEX+. What is striking in this figure is that WC-INDEX and WC-INDEX+ could achieve the same index size. The reason is that they use the same vertex ordering, and the Query-Efficient technique can only speed up the construction process, but does not have any impact on the index size. As for Naïve, its index size is the largest among these three in all datasets.

5.4.2 Query Time

Exp 3: Query Time for Road Networks. Figure 5.3 demonstrates the query time for W-BFS, Dijkstra, C-BFS, Naïve, WC-INDEX, and WC-INDEX+. An interesting obervation is that Dijkstra is the slowest among all the algorithms. It is evident from Figure 5.3 that W-BFS and C-BFS have comparable query efficiency. C-BFS is more efficient than W-BFS in terms of query time. These two BFS-based online algorithms can commit on all the datasets. The query time for the index-based technique is substantially smaller than the online search based method. On average, 4-5 orders of magnitudes speedup can be achieved. Nevertheless, the Naïve 2-hop labeling index can not be constructed for CTR and WST, hence the query time for these two datasets is set as INF. As with WC-INDEX and WC-INDEX+, they can be constructed in all datasets with a feasible index size, indexing time, and query time in microseconds. For very large road networks such as WST and CTR, the na ¨ıve indexing cannot be constructed due to memory constraint, since the method builds separate indices for each w. As a result, the query time cannot be tested and thus listed as infinity.

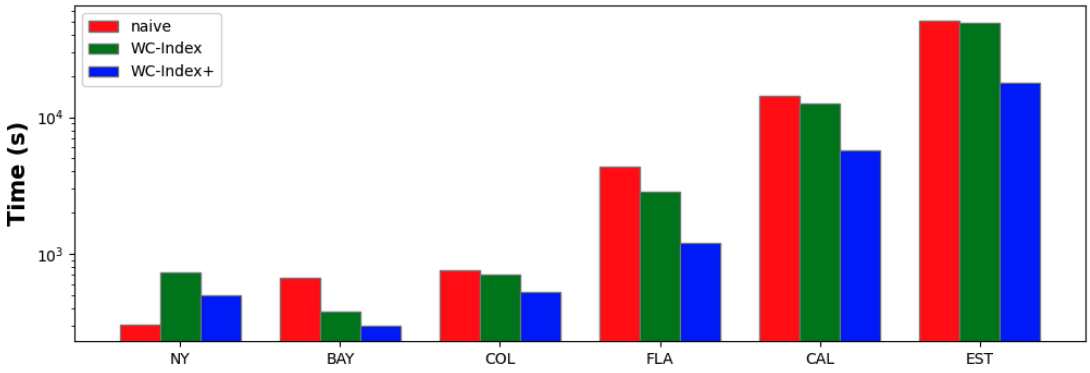


Figure 5.4: Indexing time (s) for baseline, WC-INDEX, and WC-INDEX+, when $|W| = 20$.

5.4.3 Evaluation for large $|w|$

Exp 4 investigates the indexing time and indexing size for the number of different constraint values $|w| = 20$. Figures 5.4 and 5.5 reports the findings. The results are similar to that in Exp 1 and 2. Regarding indexing time, Figure 5.4 reveals that WC-INDEX+ is the fastest method among these

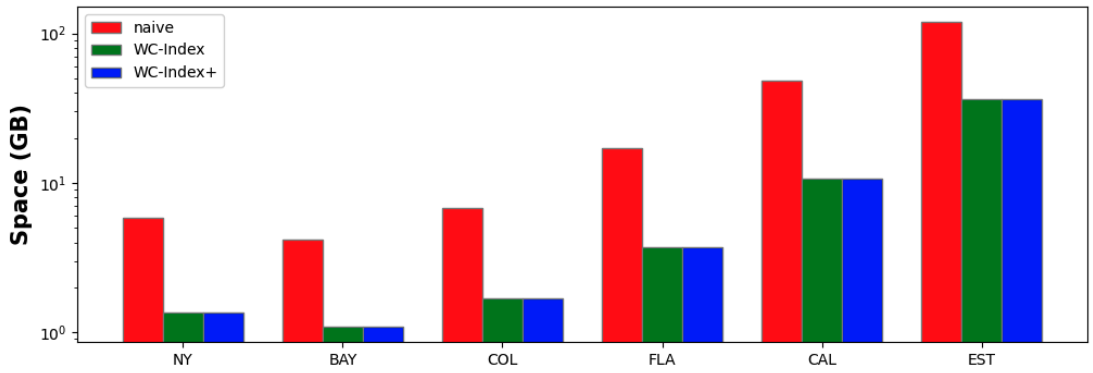


Figure 5.5: Indexing size (GB) for baseline, WC-INDEX, and WC-INDEX+, when $|W| = 20$.

three to construct the index. Regarding Naïve and WC-INDEX, WC-INDEX is slower than Naïve across all datasets evaluated, i.e., NY, BAY, COL, EST. As for indexing size, what is striking in Figure 5.5 is that WC-INDEX and WC-INDEX+ can achieve the same index size. The reason for this is because they both employ the same vertex ordering, and the Query-Efficient technique can only speed up the construction process, without affecting on the index size. As for Naïve, its index size is the largest among these three in all datasets.

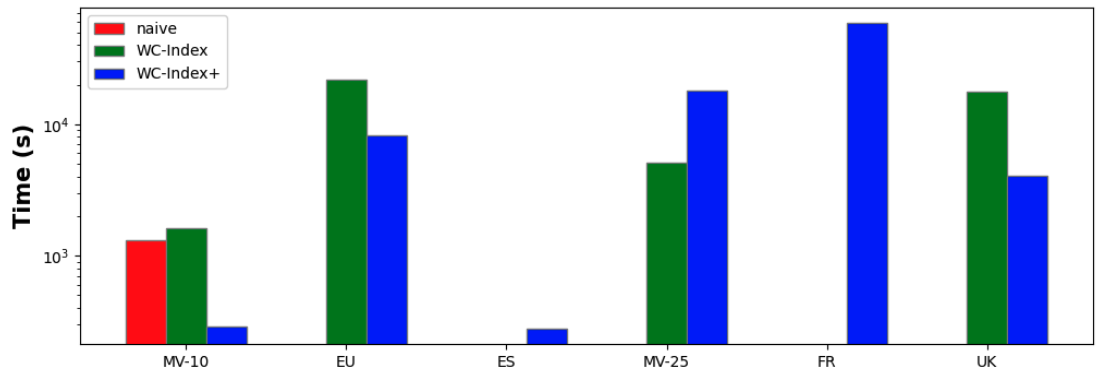


Figure 5.6: Indexing Time (s) for baseline, WC-INDEX, and WC-INDEX+.

5.5 Evaluation on Social Networks

Exp 5: Indexing Time, Size, and Query Time for Social Networks. Exp 5 evaluates the indexing time, size and query time for social networks. As shown in Figures 5.6, 5.7, and 5.8, the patterns resemble those of road networks. It is interesting to notice that the indexing time

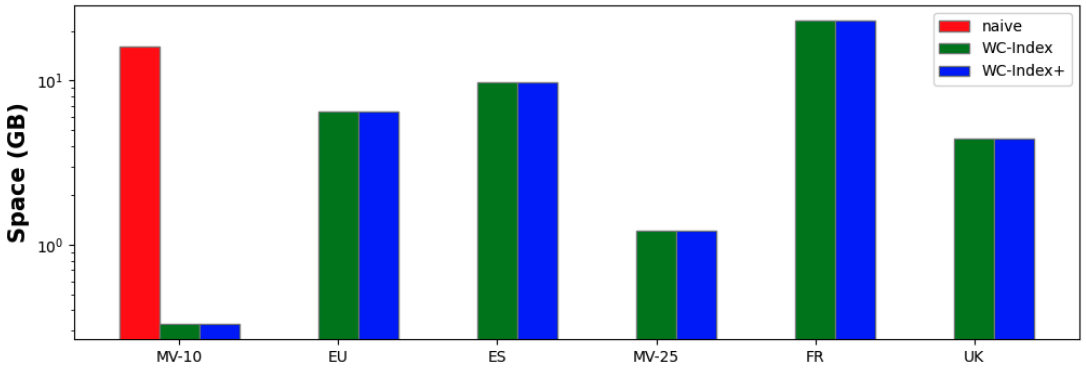


Figure 5.7: Indexing Size (GB) for baseline, WC-INDEX, and WC-INDEX+.

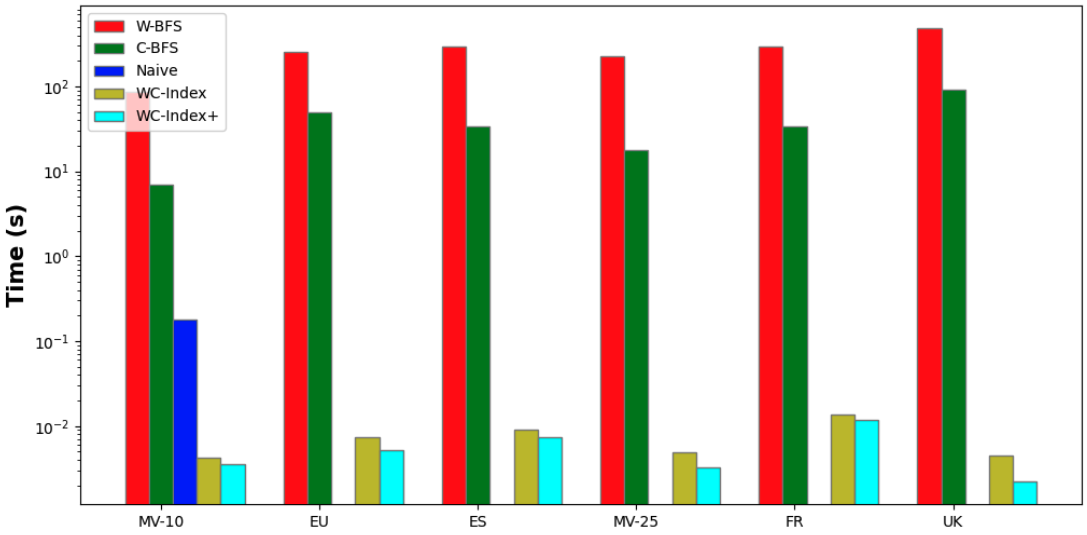


Figure 5.8: Querying time (ms) for baselines, WC-INDEX, and WC-INDEX+.

and size over social networks are larger than that of road networks since social networks have a higher average degree. For the query time, this experiment does not consider the Dijkstra since the edge is unweighted and thus it is the same as W-BFS in the social networks. The query times of WC-INDEX, and WC-INDEX+ are much faster than that of Naïve method.

Chapter 6

Conclusion and Future Directions

6.1 Conclusion and Future Works

The shortest path is a fundamental concept in graph analytics. Existing works mainly focus on the distance computer of shortest paths. Nevertheless, finding a shortest path between s and t with a quality constraint along each edge is an important problem in many applications. To bridge this research gap, this work presents a 2-hop labeling based solution to answer quality constrained shortest distance queries. Our techniques support query processing over large-scale graphs in real-time. Our approach incorporates an extension to the 2-hop index that takes advantage of the ordering of weights and distance, and significantly prunes vertices that would have been processed in a classical 2-hop index while maintaining index minimality. We also investigated the ordering of BFS searches and discovered that using vertex degree or tree decomposition can have different effects on different kinds of networks.

6.1.1 Trade-off between Space and Query Time

The proposed WC-Index possesses a relatively high space complexity compared to online methods, which seems to be an optimal solution to the Constrained Distance problem in some contexts.

For some smaller networks, the Constrained Shortest Distance problem can be easily solved by weight-constrained BFS without any extra index, as well as the corresponding extra space cost. In contrast, the proposed 2-hop labeling-based index has high space complexity. However, as presented in the experiment section, WC-Index outperforms online methods by 4-5 orders of magnitudes on larger graphs in terms of query speed. There exists a trade between index space and query time, and the usage will depend on the context of the application.

In real applications, the quality constraint shortest distance queries can be issued frequently over large-scale graphs. It requires both real-time response time and scalability. An online BFS-based search needs to traverse the graph for given query vertices s and t, making it impractical for real scenarios where real-time responses are demanded. 2-hop labeling approaches are shown to be efficient in supporting distance queries. Nevertheless, to deal with the constraints on edge qualities, a na¨ıve adaption of a 2-hop labeling solution involves constructing an index for every possible quality value w among all edges of the graph. Such a solution is infeasible since the number of distinct w values can be large. To overcome these challenges, a modified 2-hop labeling index is designed for the quality-constrained distance problem to fill this research gap. To further accelerate index construction and query processing and to reduce index size, this paper investigates various pruning methods, proposes a query-efficient approach, and develops efficient vertex ordering strategies. Since these applications deserve both real-time response time (the online method could not satisfy) and scalability (the existing index-based method could not satisfy), our proposed method could cope with all these two challenges.

The 2-hub index approach still performs much faster than conducting online constrained BFS, which is ideal in circumstances where the query response time is crucial, and queries come in frequently. The selection between the two kinds of approaches is decided by which trade-off between time and space the user would like to take. Also, since these applications deserve both real-time response time (the online method could not satisfy) and scalability (the existing index-based method could not satisfy), our proposed method could cope with all these two challenges.

6.1.2 Future Works

Maintenance under Dynamic Updates. Given a graph G, a labeling $L(\cdot)$ for G and an update ΔG to G, the problem of dynamic labeling under ΔG is to maintain $L(\cdot)$ such that the resulting labeling is the corresponding one for $G \oplus \Delta G$, where $G \oplus \Delta G$ is the graph obtained by applying ΔG to G. Since the real-world applications for our problem is naturally dynamic, it is still challenging to achieve high efficiency while maintaining the *Minimal* property.

For canonical distance labeling, the existing dynamic algorithms all fail to achieve high efficiency while retaining the minimality of the labeling [57], and it is still an open problem regarding how to address this issue. Since our labeling $L \leq (\cdot)$ contains the canonical labeling $L^c_{\leq}(\cdot)$ as a subset, the same challenge persists when designing a dynamic algorithm for $L^c_{\leq}(\cdot)$. Worse yet, the necessity to maintain the information about $\sigma_{v,w}$ can further complicate the design.

Indexing Size. As shown in the experiments, although it could speed up the indexing time by increasing the number of threads. Nevertheless, for some graphs such as DB and FL, our algorithms require far more index space than other graphs of similar size. Unfortunately, thus far it is still not clear what properties of these graphs lead to this inefficiency. Is there an efficiently computable ordering that effectively handles these graphs? Or is the huge resource consumption inevitable even with an optimal ordering under the current

References

[1] M. V. Vieira, P. B. Golgher, B. M. Fonseca, D. C. D. Reis, R. Damazio, and B. Ribeiro-Neto, "Efficient search ranking in social networks," *International Conference on Information and Knowledge Management, Proceedings*, pp. 563–572, 2007.

[2] U. Leser, "A query language for biological networks," *Bioinformatics*, vol. 21, 9 2005.

[3] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: Ranked keyword searches on graphs," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 305–316, 2007.

[4] M. Jiang, A. W. C. Fu, and R. C. W. Wong, "Exact top-k nearest keyword search in large networks," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. 2015-May, pp. 393–404, 5 2015.

[5] Y. Tao, S. Papadopoulos, C. Sheng, and K. Stefanidis, "Nearest keyword search in xml documents," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 589–600, 2011.

[6] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[7] R. Puzis, Y. Elovici, and S. Dolev, "Fast algorithm for successive computation of group betweenness centrality," *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 76, p. 056709, 11 2007. [Online]. Available: https://journals.aps.org/pre/abstract/10.1103/PhysRevE.76.056709

[8] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "A hub-based labeling algorithm for shortest paths in road networks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6630 LNCS, pp. 230–241, 2011.

[9] D. Wei, "An optimized floyd algorithm for the shortest path problem," *Journal of Networks*, vol. 5, pp. 1496–1504, 12 2010.

[10] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou, "Shortest path and distance queries on road networks: Towards bridging theory and practice," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 857–868, 2013.

[11] M. Qiao, H. Cheng, L. Qin, J. X. Yu, P. S. Yu, and L. Chang, "Computing weight constraint reachability in large networks," *The VLDB Journal 2012 22:3*, vol. 22, pp. 275–294, 8 2012. [Online]. Available: https://link.springer.com/article/10.1007/s00778-012-0288-4

[12] Q. Ma and P. Steenkiste, "Path selection for traffic with bandwidth guarantees," *International Conference on Network Protocols*, pp. 191–202, 1997.

[13] R. Xiang, J. Neville, and M. Rogati, "Modeling relationship strength in online social networks," *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, pp. 981–990, 2010.

[14] X. Zhao, J. Yuan, G. Li, X. Chen, and Z. Li, "Relationship strength estimation for online social networks with the study on facebook," *Neurocomputing*, vol. 95, pp. 89–97, 10 2012.

[15] D. Kitagawa, K. Yokota, M. Gouda, Y. Narumi, H. Ohmoto, E. Nishiwaki, K. Akita, and Y. Kirii, "Activity-based kinase profiling of approved tyrosine kinase inhibitors," *Genes to cells : devoted to molecular cellular mechanisms*, vol. 18, pp. 110–122, 2 2013. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/23279183/

[16] A. Biosa, A. Trancikova, L. Civiero, L. Glauser, L. Bubacco, E. Greggio, and D. J. Moore, "Gtpase activity regulates kinase activity and cellular phenotypes of parkinson's disease-associated lrrk2," *Human molecular genetics*, vol. 22, pp. 1140–1156, 3 2013. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/23241358/

[17] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, and Q. Zhu, "When hierarchy meets 2-hop-labeling: Effiicient shortest distance qeries on road networks," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 709–724, 5 2018.

[18] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 349–360, 4 2013. [Online]. Available: https://arxiv.org/abs/1304.4661v1

[19] A. Madkour, W. G. Aref, F. U. Rehman, M. A. Rahman, and S. Basalamah, "A survey of shortest-path algorithms," 5 2017. [Online]. Available: http://arxiv.org/abs/1705.02044

[20] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik 1959 1:1*, vol. 1, pp. 269–271, 12 1959. [Online]. Available: https://link.springer.com/article/10.1007/BF01386390

[21] S. E. Dreyfus, "An appraisal of some shortest-path algorithms," *Operations Research*, vol. 17, pp. 395–412, 6 1969.

[22] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, pp. 596–615, 7 1987. [Online]. Available: https://dl.acm.org/doi/10.1145/28869.28874

[23] M. L. Fredman and D. E. Willard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths," *Journal of Computer and System Sciences*, vol. 48, pp. 533–551, 6 1994.

[24] J. R. Driscoll, H. N. Gabow, R. Shrairman, and R. E. Tarjan, "Relaxed heaps: an alternative to fibonacci heaps with applications to parallel computation," *Communications of the ACM*, vol. 31, pp. 1343–1354, 11 1988. [Online]. Available: https://dl.acm.org/doi/10.1145/50087.50096

[25] M. Thorup, "On ram priority queues," *SODA*, pp. 59–67, 1996. [Online]. Available: http://www.diku.dk/mthorup

[26] P. van Emde Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical Systems Theory*, vol. 10, pp. 99–127, 12 1976.

[27] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," *Journal of the ACM (JACM)*, vol. 46, pp. 362–394, 5 1999. [Online]. Available: https://dl.acm.org/doi/10.1145/316542.316548

[28] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, pp. 87–90, 1958.

[29] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Mathematics*, vol. 23, pp. 309–311, 1 1978.

[30] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, and E. Gunawan, "A systematic literature review of a* pathfinding," *Procedia Computer Science*, vol. 179, pp. 507–514, 1 2021.

[31] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, p. 345, 6 1962. [Online]. Available: https://dl.acm.org/doi/10.1145/367766.368168

[32] D. V. Vliet, "Improved shortest path algorithms for transport networks," *Transportation Research*, vol. 12, pp. 7–20, 2 1978.

[33] F. Schulz, D. Wagner, and C. Zaroliagis, "Using multi-level graphs for timetable information in railway systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2409, pp. 43–59, 2002.

[34] S. Jung and S. Pramanik, "An efficient path computation model for hierarchically structured topographical road maps," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, pp. 1029–1046, 2002.

[35] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck, "Customizable route planning in road networks," *https://doi.org/10.1287/trsc.2014.0579*, vol. 51, pp. 566–591, 5 2015. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/trsc.2014.0579

[36] D. Delling, M. Kobitzsch, and R. F. Werneck, "Customizing driving directions with gpus," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial*

Intelligence and Lecture Notes in Bioinformatics), vol. 8632 LNCS, pp. 728–739, 2014. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-09873-9_61

[37] J. Arz, D. Luxen, and P. Sanders, "Transit node routing reconsidered," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7933 LNCS, pp. 55–66, 2013. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-38527-8_7

[38] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *https://doi.org/10.1137/S0097539702403098*, vol. 32, pp. 1338–1355, 2 2012. [Online]. Available: https://epubs.siam.org/doi/10.1137/S0097539702403098

[39] A. W.-C. Fu, H. Wu, J. Cheng, S. Chu, and R. C.-W. Wong, "Is-label: an independent-set based labeling scheme for point-to-point distance querying on large graphs," 11 2012. [Online]. Available: https://arxiv.org/abs/1211.2367v1

[40] M. Jiang, A. W. C. Fu, R. C. W. Wong, and Y. Xu, "Hop doubling label indexing for point-to-point distance querying on scale-free networks," *Proceedings of the VLDB Endowment*, vol. 7, pp. 1203–1214, 3 2014. [Online]. Available: https://arxiv.org/abs/1403.0779v2

[41] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Efficient query processing on spatial networks," *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pp. 200–209, 2005.

[42] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 43–54, 2008.

[43] J. Sankaranarayanan and H. Samet, "Distance oracles for spatial networks," *Proceedings - International Conference on Data Engineering*, pp. 652–663, 2009.

[44] U. Lauther, "An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background," pp. 219–230, 2004. [Online]. Available: https://www.researchgate.net/publication/230595684_An_Extremely_Fast_Exact_Algorithm_for_Finding_Shortest_Paths_in_Static_Networks_with_Geographical_Background

[45] E. Köhler, R. H. Möhring, and H. Schilling, "Acceleration of shortest path and constrained shortest path computation," *Lecture Notes in Computer Science*, vol. 3503, pp. 126–138, 2005.

[46] Z. Liu, L. Li, M. Zhang, W. Hua, P. Chao, and X. Zhou, "Efficient constrained shortest path query answering with forest hop labeling," *Proceedings - International Conference on Data Engineering*, vol. 2021-April, pp. 1763–1774, 4 2021.

[47] P. Hansen, "Bicriterion path problems," pp. 109–127, 1980. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-48782-8_9

[48] H. P. Kriegel, M. Renz, and M. Schubert, "Route skyline queries: A multi-preference path planning approach," *Proceedings - International Conference on Data Engineering*, pp. 261–272, 2010.

[49] Q. Gong, H. Cao, and P. Nagarkar, "Skyline queries constrained by multi-cost transportation networks," *Proceedings - International Conference on Data Engineering*, vol. 2019-April, pp. 926–937, 4 2019.

[50] X. Chen, H. Cai, and T. Wolf, "Multi-criteria routing in networks with path choices," *Proceedings - International Conference on Network Protocols, ICNP*, vol. 2016-March, pp. 334–344, 3 2016.

[51] C. Pornavalai, G. Chakraborty, and N. Shiratori, "Routing with multiple qos requirements for supporting multimedia applications," *Telecommunication Systems 1998 9:3*, vol. 9, pp. 357–373, 1998. [Online]. Available: https://link.springer.com/article/10.1023/A:1019160226383

[52] K. Ishida, K. Amano, and N. Kannari, "A delay-constrained least-cost path routing protocol and the synthesis method," *Proceedings - 5th International Conference on Real-Time Computing Systems and Applications, RTCSA 1998*, vol. 1998-October, pp. 58–65, 1998.

[53] S. Wang, X. Xiao, Y. Yang, and W. Lin, "Effective indexing for approximate constrained shortest path queries on large road networks," *Proceedings of the VLDB Endowment*, vol. 10, pp. 61–72, 10 2016. [Online]. Available: https://dl.acm.org/doi/10.14778/3015274.3015277

[54] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Hierarchical hub labelings for shortest paths," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7501 LNCS, pp. 24–35, 2012. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-33090-2_4

[55] T. Akiba, Y. Iwata, K.-I. Kawarabayashi, and Y. Kawata, "Fast shortest-path distance queries on road networks by pruned highway labeling," 2014.

[56] W. Li, Y. Zhang, M. Qiao, L. Chang, L. Qin, and X. Lin, "Scaling distance labeling on small-world networks," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1060–1077, 6 2019.

[57] T. Akiba, Y. Iwata, and Y. Yoshida, "Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling," *WWW 2014 - Proceedings of the 23rd International Conference on World Wide Web*, pp. 237–247, 4 2014.

[58] Y. Li, L. H. U, M. L. Yiu, and N. M. Kou, "An experimental study on hub labeling based shortest path algorithms," *Proceedings of the VLDB Endowment*, vol. 11, pp. 445–457, 12 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3186728.3164141

[59] R. Halin, "S-functions for graphs," *Journal of Geometry 1976 8:1*, vol. 8, pp. 171–186, 1976. [Online]. Available: https://link.springer.com/article/10.1007/BF01917434

[60] N. Robertson and P. D. Seymour, "Graph minors. iii. planar tree-width," *Journal of Combinatorial Theory, Series B*, vol. 36, pp. 49–64, 2 1984.

[61] H. L. Bodlaender, "A tourist guide through treewidth," *Acta Cybernetica*, vol. 11, 1993.

[62] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM Journal on Algebraic Discrete Methods*, vol. 8, pp. 277–284, 4 1987. [Online]. Available: https://dl.acm.org/doi/10.1137/0608024

[63] A. Berry, P. Heggernes, and G. Simonet, "The minimum degree heuristic and the minimal triangulation process," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2880, pp. 58–70, 2003. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-39890-5_6

[64] J. Kunegis, “Konect,” pp. 1343–1350, 5 2013.

[65] L. J, “Stanford large network dataset collection,” 2011. [Online]. Available: https://snap.stanford.edu/data/