FHL-Cube: Multi-Constraint Shortest Path Querying with Flexible Combination of Constraints

Ziyi Liu¹, Lei Li^{2,3,*}, Mengxuan Zhang⁴, Wen Hua¹, Xiaofang Zhou^{3,2}

¹School of ITEE, The University of Queensland, Brisbane, QLD, Australia

²DSA Thrust, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China

³Department of CSE, The Hong Kong University of Science and Technology, Hong Kong SAR, China

⁴Department of Electrical and Computer Engineering, Iowa State University, USA

ziyi.liu@uq.edu.au,thorli@ust.hk,mxzhang@iastate.edu,w.hua@uq.edu.au,zxf@cse.ust.hk

ABSTRACT

Multi-Constraint Shortest Path (MCSP) generalizes the classic shortest path from single to multiple criteria such that more personalized needs can be satisfied. However, MCSP query is essentially a highdimensional skyline problem and thus time-consuming to answer. Although the current Forest Hop Labeling (FHL) index can answer MCSP efficiently, it takes a long time to construct and lacks the flexibility to handle arbitrary criteria combinations. In this paper, we propose a skyline-cube-based FHL index that can handle the flexible MCSP efficiently. Firstly, we analyze the relation between low and high-dimensional skyline paths theoretically and use a cube to organize them hierarchically. After that, we propose methods to derive the high-dimensional path from the lower ones, which can adapt to the flexible scenario naturally and reduce the expensive high dimensional path concatenation. Then we introduce efficient methods for both single and multi-hop cube concatenations and propose pruning methods to further alleviate the computation. Finally, we improve the FHL structure with lower height for faster construction and query. Experiments on real-life road networks demonstrate the superiority of our method over the state-of-the-art.

PVLDB Reference Format:

Ziyi Liu, Lei Li, Mengxuan Zhang, Wen Hua, Xiaofang Zhou. FHL-Cube: Multi-Constraint Shortest Path Querying with Flexible Combination of Constraints. PVLDB, 15(11): 3112-3125, 2022. doi:10.14778/3551793.3551856

1 INTRODUCTION

Multi-criteria decision-making on path-finding is important in our daily life due to the increasingly informative road networks that can satisfy various user demands. For example, a user may have a limited budget to pay the toll charge of highways, bridges, tunnels, or congestion. Some mega-cities require drivers to reduce the number of big turns (e.g., left turns in the right driving case [16] as they have a higher chance to cause accidents). There are other side-criteria such as the minimum height of tunnels, the maximum capacity of roads, the maximum gradient of slopes, the total elevation increase, the length of tourist drives, the number of transportation changes,

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097. doi:10.14778/3551793.3551856

* Lei Li is the corresponding author.

etc. However, most of the existing path-finding algorithms only optimize one objective and ignore the other side-criteria, such as minimum distance [10, 32, 64], travel time of driving [30, 31, 62] or public transportation [55, 58], fuel consumption [29, 33], battery usage [3], etc. In fact, it is these criteria and their combinations that endow different routes with diversified meaning and satisfy users' flexible needs. For example, users' requirements on pathfinding may change according to their current events or the means of transport used: A commuter usually likes the most fuel-efficient path within a limited time budget; a cyclist or walker may prefer the shortest while labor-saving (gentle slope) way; a traveler prefers the most attractive path within limited tolls. Hence, this flexible multicriteria route planning is more generalized while the shortest path is its special case. Moreover, it is also an important research problem in the fields of both transportation [22, 51] and communication (Quality-of-Service constraint routing) [8, 25, 53, 54].

Motivations. Typically, skyline path [15, 27, 44] is applied to find the result which is optimal in every criterion when there are multiple optimization goals. It provides a set of paths that cannot dominate each other in all criteria. However, skyline path computation is very time-consuming with time complexity $O(c_{max}^{n-1} \times |V| \times$ $(|V| \log |V| + |E|))$, where |V| is the vertex number, |E| is the edge number, c_{max} is the largest criteria value, and n is the number of criteria [18] (c_{max}^{n-1} is the worst-case skyline number). In addition, it is impractical to provide users with a large set of potential paths for them to choose from. Therefore, the Multi-Constraint Shortest Path (MCSP) is widely applied and studied. Specifically, it finds the best path based on one objective while requiring other criteria satisfying some predefined constraints. For example, suppose each road has a distance w and two costs c_1 , c_2 , and we set the maximum constraints C_1 and C_2 on the costs. Then this *MCSP* problem finds the shortest path *p* whose cost $c_1(p) \leq C_1$ and $c_2(p) \leq C_2$.

Nevertheless, it is non-trivial to solve the flexible *MCSP* problem where users could specify arbitrary constraint combinations based on their current requirements. Firstly, it inherits all the challenges from the skyline path search as proved in [36]. In fact, its simplest version *Constraint Shortest Path (CSP)* (with only one constraint) is already an NP-H problem [12, 19], and this is why most of the existing works only focus on *CSP* queries and many of them trade the result optimality for efficiency [18, 19, 23, 38, 53, 56]. Currently, *Forest Hop Labeling (FHL)* [36] is the only *CSP* algorithm that can achieve both accurate and efficient results. However, its *MCSP* version [37] takes orders of magnitude longer time to construct the path index because both the skyline path number and the skyline validation cost soar up as the dimension increases. Moreover, *FHL* is

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

only efficient for the *MCSP* queries with the same constraint number, but its performance deteriorates when the query's constraint is a subset due to its huge label redundancy and limited pruning power. Another way to adapt to the flexibility is building indexes for each constraint combination (Figure 1-(a) *FHL-Multi*), but it would take a longer time to construct and result in a much larger index space. Instead, in this work, we aim to design a single index (Figure 1-(b) *FHL-Cube*) to support flexible *MCSP* queries. Detailed analysis of these approaches is provided in Section 3.



(b) FHL - Cube: One index with each label contains a cube of all combination

Figure 1: FHL and FHL-Cube Comparison

Challenges. The flexible MCSP requires answering queries of arbitrary criteria combinations, so the first challenge lies in how to build an index to support the query flexibility. Even though the higher dimensional MCSP queries take larger index space with longer construction and query time than the lower ones, it is observed that the query results of the low dimension also belong to those of the high dimension. This insight provides us with a chance to derive the high dimension skyline paths by accumulating those from the low dimension. To this end, it is critical to distinguish between low dimensional skyline paths and high dimensional ones and establish their relations. Specifically, we first propose the ex*clusive skyline path* to determine the lowest dimension of a skyline path, such that the skyline paths can be organized hierarchically into a cube by categorizing them into different subspaces. Then, we extend FHL to FHL-Cube by changing the index values from paths of only one space to a cube that organizes all the spaces as shown in Figure 1-(b). Next, we establish their relations and propose theorems to derive high dimensional results from lower ones. Finally, the higher dimensional cuboid (the basic component of cube) is smaller than the lower ones (reverse to the FHL as indicated by the database sizes), and the flexible MCSP can be handled naturally.

In addition, the cuberization of skyline paths involves cube concatenation in both index construction and query processing, which would cause numerous intermediate skyline results with heavy computation. The difficulty here is to improve the cube concatenation efficiency such that the index construction and query processing can be accelerated as well. Naively, we concatenate the skylines of the corresponding cuboids from two-dimension to higher dimensions, since the subspaces are organized hierarchically. However, there are multiple two-dimension subspaces waiting for concatenation. Hence, we dedicate to prune the unnecessary computation in cube concatenation through subspace pruning and hop pruning, respectively. To be specific, we determine the path range of each subspace through calculating the lower and upper bounds of each dimension, such that the subspaces whose path range is dominated by others could be safely pruned without affecting the correctness. In terms of the hop pruning, we precompute the lower bound and the upper bound of the path results concatenated by each hop and those hops with their lower bound surpassing others' upper bound can be safely pruned. Moreover, it is proved that those pruned subspaces or pruned hops of one specific dimension also applies for all its higher dimensions. Finally, both the index construction and query processing of *FHL-cube* is highly improved owing to the proposed pruning techniques of cube concatenation.

Finally, it is observed that the tree height of the *tree decomposition* structure during the index construction is usually large, which seriously deteriorates the index performance since larger tree height indicates more cube concatenation in both index construction and query processing. To alleviate this problem, we analyze the forest structure and propose several principles to optimize the index structure. As for the query processing, our index is flexible for any combination of criteria because of the skyline path relation established between low dimension and high dimension.

Contributions. 1) We introduce a novel flexible *MCSP* problem and propose the *FHL-Cube* index for its query processing; 2) We achieve the query flexibility by organizing and deriving paths with different dimensions hierarchically as a cube, and further propose pruning techniques for faster cube concatenation; 3) We optimize the boundary label structure and concatenation method for faster index construction and querying; 4) We conduct extensive evaluations on real-world road networks to verify the superiority of our approach compared with the state-of-the-art methods.

2 PRELIMINARY

2.1 **Problem Definition**

A multi-criteria road network is a *n*-dimensional graph G(V, E), where *V* is a vertex set and $E \subseteq V \times V$ is an edge set. Each $e \in E$ has *n* criteria falling into two categories: a *weight* w(e) and a set of *costs* $\{c_i(e)|i \in [1, n-1]\}$. A path *p* from $s \in V$ to $t \in V$ is a sequence of consecutive vertices $p = \langle s = v_0, v_1, \dots, v_k = t \rangle = \langle e_0, e_1, \dots, e_{k-1} \rangle$, where $e_i = (v_i, v_{i+1}) \in E$, $\forall i \in [0, k-1]$. Each path *p* has a weight $w(p) = \sum_{e \in p} w(e)$ and a set of costs $\{c_i(p) = \sum_{e \in p} c_i(e)\}$. We assume the graph is undirected and will extend to directed in Section 6.4. We denote a *n*-dimensional space as $D^n = \{w, c_1, ..., c_{n-1}\}$ to map each criterion in *G*. In addition, we use β to represent a combination of criteria for short. We define the *MCSP* queries as follows:

DEFINITION 1 (CONSTRAINT GROUP). A constraint group C^{β} is a set of values $\{C_i\}$, where $C_i \in \mathbb{R}^+$, $i \in [1, n - 1]$. The size of the group is $|\beta| \in [1, n - 1]$, and $C^{|\beta|}$ denotes all the groups with size $|\beta|$.

For example, given a 4-dimensional graph, constraint group $C^{\{C_1,C_3\}}$ belongs to C^2 , and $C^{\{C_1,C_3,C_4\}}$ is another one that belongs

to C^3 . For the ease of explanation, the notation of constraint group is simplified, i.e., $C^{\{C_1, C_3, C_4\}}$ is written as $C^{C_1C_3C_4}$

DEFINITION 2 (**FLEXIBLE MCSP QUERY**). Given a n-dimensional graph G(V, E), a MCSP query $q(s, t, C^{\beta})$ returns an optimal path with the minimum w(p) while $c_i(p) \leq C_i, \forall C_i \in C^{\beta}$.

Note that β can be varying and it involves $\sum_{i=1}^{n-1} \binom{n-1}{i}$ different constraint groups. Thus, our solution will support queries from all these groups, which highlights the flavor of the "flexibility" of *MCSP* query answering. In the following, we also define *fixed MCSP* query as a reference:

DEFINITION 3 (**FIXED MCSP QUERY**). Given a n-dimensional graph G(V, E) and a fixed constraint group \overline{C} , a fixed MCSP query $q(s, t, \overline{C})$ returns an optimal path with the minimum w(p) while $c_i(p) \leq C_i, \forall C_i \in \overline{C}$.

For brevity, the term "*MCSP* query" discussed in the remaining paper refers to *flexible MCSP* query. As analyzed in Section 1, it is impractical to answer a *MCSP* query by online graph search. Therefore, in this paper, we resort to index-based method and study the following problem:

DEFINITION 4 (INDEX-BASED MCSP QUERY PROCESSING). Given a n-dimensional graph G(V, E), we aim to construct a labelbased path index L such that any flexible MCSP query $q(s, t, C^{\beta})$ in G can be answered efficiently only with L.

We assign each vertex $v \in V$ a label $L(v) \in L$. Specifically, L(v) contains the set of all the path index starting from v and we use $L(v, u) \in L(v)$ to denote the set of path index from v to u. Unlike the shortest path index which stores the shortest distance from one vertex to another, the *MCSP* index needs to cover all the possible constraints of each criterion C_i and any constraints' combination C^{β} . In addition, we observe that the *MCSP* paths are essentially a subset of the *multi-dimensional skyline* paths as introduced below.

2.2 Multi-Dimensional Skyline Path

Like the *dominance* relation of the skyline query in the 2-dimensional space [46], we first define the *dominance* relation between any two paths with the same source and destination vertices:

DEFINITION 5 (*n*-**DIMENSIONAL PATH DOMINANCE**). Given two paths p_1 and p_2 with the same *s* and *t*, p_1 dominates p_2 in D^n iff $w(p_1) \le w(p_2)$ and $c_i(p_1) \le c_i(p_2), \forall i \in [1, n - 1]$, and at least one of them is the strictly smaller relation.



Figure 2: An Example Road Network

DEFINITION 6 (*n*-**DIMENSIONAL SKYLINE PATH**). Given two paths p_1 and p_2 with the same *s* and *t*, they are *n*-dimensional skyline paths if they cannot dominate each other in D^n .

We use P(s, t) to denote the set of skyline paths from *s* to *t*. Let us take the sub-graph G_1 shown in Figure 2-(b) as an example. We assume that each edge contains four criteria. We can enumerate five paths from v_1 to v_6 : $p_1 = (3, 2, 7, 7)$; $p_2 = (5, 5, 5, 4)$; $p_3 = (7, 3, 4, 8)$; $p_4 = (7, 4, 6, 3)$; $p_5 = (5, 5, 11, 4)$.

 p_2 dominates p_5 so p_5 is not a skyline path. Meanwhile, as the other four paths cannot dominate each other, we obtain four 4-dimensional skyline paths: $P(v_1, v_6) = \{p_1, p_2, p_3, p_4\}$. Next we prove that skyline paths are essential for building any *MCSP* index:

DEFINITION 7 (SUBSPACE PATH). D^{β} is a sub-dimensional space of D^{n} , where $|\beta| \leq n$ is the number of the dimensions in D^{β} . $\beta(p)$ is the projection of path p in D^{β} .

DEFINITION 8 (SUBSPACE SKYLINE PATH). Given two paths p_1 and p_2 with the same s and t, their projections $\beta(p_1)$ and $\beta(p_2)$ are subspace skyline path of D^{β} if they cannot dominate each other.

THEOREM 1. The skyline paths between any two vertices are complete and minimal for all the possible MCSP queries.

PROOF. We first prove the completeness. Suppose the skyline paths from *s* to *t* are $\{p_1, \ldots, p_k\}$. Each cost dimension c_i can be divided into k + 1 intervals: $[0, c_i^1), [c_i^1, c_i^2), \ldots, [c_i^k, \infty)$, where each c_i^j is the j^{th} cost value of criterion c_i sorted increasingly, and the entire space can be decomposed into $(k + 1)^{(n-1)}$ subspaces. Then for the subspaces with any interval falling into $[0, c_i^1)$, there is no valid path satisfying all the constraints at the same time. For the subspaces with no interval falling into $[0, c_i^1)$, there is always at least one path dominating all the constraints in it, and the one with the smallest weight is the result.

Next we prove this skyline path set is minimal. Suppose we remove any skyline path $p_j = \{c_i^j\}$ from the skyline result set. Then the *MCSP* query whose constraint is within the subspace of $[c_1^j, c_1^{j+1}) \times [c_2^j, c_2^{j+1}) \times \cdots \times [c_k^j, c_k^{j+1})$ will have no valid path. Hence, p_j cannot be removed and the skyline path set is minimal for all the *MCSP* queries between *s* and *t*.

In summary, building an index for *MCSP* query is equivalent to building the index for multi-dimensional skyline path query.



Figure 3: FHL Example

2.3 Forest Hop Labeling (FHL)

We briefly describe the FHL structure since it is the foundation of our FHL-Cube. To reduce the large index size introduced by long skyline paths, it first partitions G into a set $\{G_1, \ldots, G_k\}$ of vertex-disjoint subgraphs such that $\bigcup_{i \in [1,k]} V(G_i) = V(G), V(G_i) \cap V(G_j) =$ $\emptyset(\forall i \neq j, i, j \in [1, k])$. For example, Figure 3-(a) partitions Figure 2-(a) into $\{G_1, G_2, G_3, G_4\}$. If an edge connects two different subgraphs, it is a boundary edge and its end vertex is called boundary vertex, such as the dashed edges in Figure 3-(a). Moreover, FHL extracts all the boundary edges with their vertices to form a boundary graph, such as Figure 3-(b). Note that FHL pre-computes the skylines between each pair of boundary vertices in every partition to ensure the correctness of index construction, and adds shortcuts for them if they are not connected in the original graph (the red edges). After that, it maps each subgraph into a small tree structure by tree decomposition. Similarly, the boundaries of the small trees form boundary tree. The small trees and the boundary tree together form a forest. The tree decomposition is introduced below.

DEFINITION 9 (**TREE DECOMPOSITION**). Given a graph G, its tree decomposition T_G is a rooted tree in which each node X_i is a subset of V. It has the following tree properties: 1) $\bigcup X_i = V$; 2) $\forall (u, v) \in E, \exists X_i \text{ such that } \{u, v\} \subseteq X_i; 3) \forall v \in V$, the set $\{X_i | v \in X_i\}$ forms a subtree of T_G .



Figure 4: Tree Decomposition and Bottom-Up Tree

For example in Figure 4-(a), the first vertex in each tree node is its representative vertex. The index labels are assigned from a vertex to all its ancestors in T_G . For example, $L(v_4)$ contains labels from v_4 to v_1, v_3 , and v_6 . Besides, this structure has a *Cut Property* that can be used to answer queries: $\forall s, t \in V$ and X_s, X_t are their corresponding tree nodes without an ancestor/descendant relation, their cuts are in their *Lowest Common Ancestor (LCA)* node [6]. For example, the tree nodes of v_2 and v_4 are X_2 and X_4 , and their *LCA* is $X_1 = \{v_1, v_3\}$. Then these two vertices form a cut set between v_2 and v_4 , and the skyline results are *Skyline*($\{L(v_2, v_i) \oplus L(v_i, v_4)\}$), $\forall v_i \in X_1$, where \oplus is the concatenation operator of two skyline paths. For the queries with *s* and *t* on different small trees in the forest, the *MCSP* results are obtained with the help of the boundary tree.

3 ANALYSIS OF FHL ON FLEXIBLE MCSP

For a *n*-D space, there could be $\sum_{i=2}^{n} {n \choose i}$ subspaces. Figure 4-(b) shows a bottom-up enumeration tree for a 4D space $wc_1c_2c_3$, and each *k*-D ($2 \le k \le n$) space can be divided into C_k^2 (k - 1)-D subspaces. For instance, the 3D space wc_1c_2 can be divided into three 2D subspaces: wc_1 , wc_2 , and c_1c_2 . In this section, we analyze the two *FHL* approaches for the flexible *MCSP*. We call the one with index only on the highest dimension *FHL* [37], and the

one with index on every subspace *FHL-Multi*. Specifcially, because *FHL-Multi* has the fastest query performance, we analyze its index construction; because *FHL* has the smallest index, we analyze its query performance.

1) Hardness of the High-Dimensional Skyline Path Concatenation. Given two skyline path sets $P(v, h_i)$ and $P(h_i, u)$ with sizes of m and n, where $\{h_i\}$ is the set of v and u's common hops (i.e., cut set), we can compute the d-D skyline paths $P(v, h_i, u)$ from v to u via each h_i , which is the skyline results from $P(v, h_i) \oplus P(h_i, u)$, with the time complexity $O(Nlog^{max(1,d-2)}N)$, where d is the dimension number and N = mn is the cardinality of generated data. It should be noted that this time complexity does not just have a higher order of log as it seems, the skyline paths sizes m and n also increase exponentially as the dimension number grows. Moreover, as u and v have multiple hops, it takes $O(\sum_{i=1}^{k} |P(v, h_i, u)| \log \sum_{i=1}^{k} |P(v, h_i, u)|)$ time in total to obtain the final skylines P(v, u).

2) FHL Query Processing. To answer a query $q(s, t, C^{\beta})$, we first compute the *LCA* X of X(s) and X(t). We can compute the subspace skyline paths P(s, t) of $D^{\beta \cup \{w\}}$ through the concatenation: $P(s, h_i) \oplus P(h_i, t), \forall h_i \in X$ in $O(Nlog^{max(1, |\beta| - 1)}N)$ time. After that, we find the one with minimum w under the constraints of C^{β} in constant time. If the query points are from different partitions, similar process would occur three times (in the small trees and boundary tree respectively). However, N here comes from the highest dimensional space so it is much larger than the ones actually needed. To make things worse, the pruning power among multiple hops [37] also decreases due to high-overlapping in the high dimensional space. So the efficiency using *FHL* for subspace.

3) FHL-Multi Index Construction. To adapt FHL to FHL-Multi, its *n*-D skyline paths are replaced with all their subspace skyline paths based on the enumeration tree as shown in Figure 1-(a). During the index construction, to compute a label from *v* to each *u*, we consider all the vertices in $X_v \setminus \{v\}$ as their common hops. The skyline paths are concatenated in each D^β instead of one fixed *d*-D, so time complexity of the concatenation on each hop becomes $O(\sum_{d=2}^n {N \choose d} \cdot Nlog^{max(1,d-2)}N)$, and the total complexity on *k* hops takes $O(\sum_{d=2}^n {n \choose d} \cdot \sum_{i=1}^k |P(v, h_i, u)| \log \sum_{i=1}^k |P(v, h_i, u)|)$ time. Besides, the index sizes also grows exponentially, which makes this approach impractical for real-life applications.

In summary, fast query processing requires a huge index (*FHL-Multi*), while a reasonable-size index (*FHL*) is not efficient enough for query processing. Therefore, we describe our *FHL-Cube* that is query efficient while having a reasonable index size.

4 FHL-CUBE CONSTRUCTION

FHL-Cube consists of two levels of index: a set of inner skylinecube trees for each partition and a boundary skyline-cube tree to organize the inner trees. Because these two structures follow the similar construction procedure, we describe how to construct the skyline-cube tree first. Then we present how to further optimize the boundary tree construction based on the problems in *FHL*.

4.1 Skyline Cube Tree Construction

Before constructing the index, we first compute the all-pair skyline results of each partition's boundaries on the original graph such that the path information detouring outside each partition is also captured. After that, we add this boundary all-pair information to each partition and start the index construction. The construction is made up of three steps as shown in Algorithm 1. The first step (lines 2-6) forms the tree nodes by contracting the vertices in order (we use the Minimum Degree Elimination [15, 34]), with the boundary vertices the last to contract. We denote the vertex set including v and it neighbor set N(v) at the time when v is contracted as X_v . The contraction utilizes the single hop skyline cube concatenation. Because the edges between the boundaries are already computed, we can use the edge information directly without actual concatenation. Secondly, a tree is formed by setting the parent of each X_v as X_u with $u \in X_v$ the lowest-ranking vertex in X_v (lines 8-9). Finally, the labels are populated from the tree root in a depth-first fashion by reusing the labeling of its ancestors' (lines 11-15). Specifically, we assign labels to v from X_v to all its ancestors X_u , because the vertices in X_v naturally form a cut between v and all its ancestors. Then for any vertex pair (s, t) with X_v as the lowest common ancestor of X_s and X_t , we can view the vertices in X_v as their hops. We use multi-hop cube concatenation to obtain the label cubes. Details about the skyline cubes and cube concatenation will be introduced in Section 5 and Section 6 respectively. Due to limited paper space, we put the example index in [2].

Algorithm 1: FHL Cube Construction									
Input: Graph $G(V, E)$									
Output: FHL-Cube Labeling L									
1 // Tree Node Contraction									
2 while $v \in V$ has the minimum degree and $V \neq \phi$ do									
$X_v = N(v), r(v) \leftarrow \text{Iteration Number}$									
for $(x, y) \leftarrow N(v) \times N(v)$ do									
$Cube_{x,y} \leftarrow Cube(Cube_{x,v} \oplus Cube_{v,y} \cup Cube_{x,y});$									
$E \leftarrow E \cup (P(x, y)), V \leftarrow V - v, E \leftarrow E - (x, v) - (v, y);$									
// Tree Formation									
for v in $r(v)$ increasing order do									
$u \leftarrow min\{r(u) u \in X(v)\}, X_v.Parent \leftarrow X_u$									
// Label Assignment									
1 $X_v \leftarrow \text{Tree Root}, Q.insert(X_v)$									
while $X_v \leftarrow Q.pop()$ do									
for $u \in \{u X_u \text{ is ancestor of } X_v\}$ do									
$P(v, u) \leftarrow Cube(\bigcup Cube_{v,h} \oplus Cube_{h,u}) \forall h \in X_v);$									
$L(v) \leftarrow (u, Cube_{v,u});$									
$Q.insert(X_u.children), L \leftarrow L \cup L(v);$									

4.2 Boundary Tree Construction

The boundary graph is made up of all partition boundaries. Same as *FHL*, we can contract the boundaries in the unit of partition with the same order used in the inner-partition index to keep the tree structure of the boundaries from the same partition stable. Moreover, all the concatenations used between the boundaries from the same partition can also be skipped as we have already obtained their cubes. However, this contraction method still takes a very long time to propagate on the boundary tree, which includes a series of skyline path cube concatenations among the tree nodes. This is because *FHL* constructs the boundary tree following the contracting order of the small trees without considering the hierarchical structure among the partitions. This makes the boundary tree become very unbalance and the tree height increases dramatically. Moreover, as discussed in Section 2.3, when we create a boundary graph, we first

build a complete graph for each partition and then connect them together by all cutting edges. Therefore, it results in a very dense



Figure 5: Contracting Orders with Different Principles

Hence, we propose a contraction method on the boundary graph to achieve the balance of the boundary tree and then reduce the computation on the propagation phase. To ensure the hierarchical structure of the partitions, we first create a connected graph that presents their connective relationships. Specifically, we treat each partition as a node and add an edge between two nodes if their corresponding partitions are connected in the original boundary graph. For example, Figure 5 is the connected graph of Figure 3-(b). It displays the procedure of contracting in two different orders. Even though both Figure 5-(a) and Figure 5-(b) contract the graph by the order of vertex degree, the contraction in Figure 5-(b) is better than Figure 5-(a), because it can produce lower tree height. As a consequence, we propose our observation as follows:

LEMMA 2 (SINGLE CHILD TREE DECOMPOSITION (SC-TD)). If the vertices in a contracting order are consecutive and form a path, then each node in its tree decomposition only has a single child.

PROOF. When contracting each vertex v, it will form a tree node, including v and its current neighbors. After contracting all vertices, we check the vertices of each node X_v , except v, to find the vertex u contracted prior to others, and select its corresponding node X_u as the parent of X_v . As the vertices in the contracting order are consecutive, the next contracting vertex must be a neighbor of the current contracting vertex. Therefore, the corresponding node of the next contracting vertex will always be the child of the current contracting one. This property degenerates the tree into a stick.

Note that the *SC-TD* or the extremely unbalanced tree is very common in dense graphs, such as the connected graph of New York City, where most of its partitions connect to many other partitions. Moreover, when the graph is complete, its tree decomposition must be a *SC-TD*. Therefore, it is critical to solve this contracting problem in the boundary graph. Based on Lemma 2, we aim to break the consecutive of the contraction order: if several vertices following the consecutive order can form a path on the graph, we reduce its length as much as possible. Therefore, we iteratively choose a vertex to contract with the following principles: 1) smallest degree; 2) if *v* is the neighbor of the last contracted vertex, we find the next one to extract; 3) the number of times each vertex *v* appears as a neighbor is recorded as Count(v); 4) if 2) cannot be held, we contract the vertex *v* with the smallest Count(v).

5 SKYLINE PATH RELATIONS AND CUBE

Intuitively, concatenation in lower dimensions is much faster, so we resort to "push it down" from higher to lower dimensions. Firstly, we discuss the relationship between the lower-dimensional and higher-dimensional skyline paths. Then we propose several observations to derive the higher dimensional skyline path from the lower ones. Finally, we present the *skyline path cube* that organizes the skyline paths into different categories, which will further improve the concatenation efficiency and support flexibility.

5.1 Skyline Path Categorization

Because two paths with the same value in one subspace may have different values in other dimensions, we first distinguish the basic types of non-dominated paths based on their values:

DEFINITION 10 (NON-DOMINATED PATH TYPE). Given two non-dominated paths p and q, they are **Indistinct** if they have the same values on all dimensions; otherwise they are **Incomparable**.

For example, p = (1, 1, 2, 3) and q = (1, 1, 3, 2) are *incomparable*, while p = (1, 1, 2, 3) and q = (1, 1, 2, 3) are *indistinct*. Although two paths are indistinct in 4D, they could have different values on other dimensions. Therefore, we further discuss the path relations within a specific subspace as follows:

DEFINITION 11 (**SKYLINE SUBSPACE**). Given a set P_s of skyline paths in subspace D^{β} ($\beta \in [2, n]$), we say D^{β} is: an **Incomparable Subspace** $D^{\beta}_{\lambda}\{p\}$ of p if $\beta(p)$ is incomparable with the projections of other skyline paths in D^{β} ; an **Indistinct Subspace** $D^{\beta}_{=}\{p\}$ of p if $\exists q \in P_s, \beta(p)$ and $\beta(q)$ are indistinct in D^{β} ; a **Dominated Subspace** $D^{\beta}_{<}\{p\}$ of p if $\exists q \in P_s$ dominates $\beta(p)$ in D^{β} .

For example in Figure 6-(a), $P_s = \{p_1, p_3, p_5\}$ is a skyline path set in subspace wc_1 , and D^{wc_1} is 1) $D^{\beta}_{\lambda}\{p_3\}$ an incomparable subspace for p_3 because it has different values with p_1 and p_5 ; 2) $D^{\beta}_{=}\{p_1, p_5\}$ an indistinct subspace as the their path values are the same in wc_1 ; 3) $D^{\beta}_{<}\{p_2, p_6\}$ dominated, because their projection (3, 7) and (3,5) in wc_1 are dominated by $wc_1(p_1) = (2, 4)$.



Figure 6: 4D Path Example

Depending on the subspace types, we divide the *n*-D paths into the following four types: **Case I Path** has at least one incomparable subspace; **Case II Path** only has indistinct subspaces; **Case III Path** has indistinct and dominated subspaces; **Case IV Path** only has dominated subspaces.

Next, we will present the skyline path derivation using the categorization above.

5.2 Skyline Path Derivation

To establish the relations between the lower dimensions and the higher dimension, we discuss the properties of these four path categories. We first have the following theorem for the *Case I Path*:

THEOREM 3 (INCOMPARABLE LOWER DOMINATES HIGHER). p is a skyline path in D^n if it has an incomparable subspace $D^\beta \subseteq D^n$.

PROOF. Suppose D^{β} is p's incomparable subspace. If p is not a skyline path in D^n , then $\exists p'$ dominates p in D^n , which dominates p in D^{β} . This contradicts with the incomparable definition.

For example, given a 4D path $p(wc_1c_2c_3)$, if it is a skyline path in wc_1 and $D^{\{wc_1\}}$ is its 2D incomparable subspace, then p must be a skyline path in all the superspaces of $D^{\{wc_1\}}$, including $D^{\{wc_1c_2\}}$, $D^{\{wc_1c_3\}}$, and $D^{\{wc_1c_2c_3\}}$. When we generate the skyline paths to higher dimensions, we can obtain the *Case I Paths* on the fly. Moreover, the number of *Case I Paths* keeps monotonically increasing when the dimension increases, because the emerging subspaces could be new incomparable subspaces, which can produce more skyline paths. For *Case II Paths*, we have the following theorem.

THEOREM 4 (FULL INDISTINCT DOMINATES HIGHER). p is a skyline path in D^n if any D^2 is an indistinct subspace of p.

PROOF. We also prove it by contradiction. Suppose p is indistinct in all D^2 but not a skyline path in D^n , then $\exists p'$ dominates p in D^n . Therefore, there exists at least one dimension c_i such that $p'(c_i) < p(c_i)$ while others are smaller or equal, so p could not be indistinct in those subspace D^2 containing c_i .

For *Case III Paths*, we have the following theorem.

THEOREM 5 (**PARTIAL INDISTINCT AND INCOMPARABLE**). $\forall q$ that is indistinct skyline path of p in some D^2 subspaces, p is a skyline path if it cannot be dominated by all these q in other D^2 subspaces.

PROOF. If any indistinct *q* can dominate *p* in all the other spaces, then *q* can dominate *p* in D^n , so *p* is not a skyline.

As for the *Case IV*, it is a little tricky because it only has dominated subspaces and it seems cannot be a skyline path. However, this is not correct. For example in Figure 6-(a), $\overline{p_6}$ is dominated in all the D^2 subspaces, but it cannot be dominated in the D^4 space. Therefore, a path can be a skyline path even though it is dominated in every D^2 . This is caused by the fact that the skyline paths within different dimensions cannot hold monotonicity [48]. The following theorems help to rule out some non-skyline paths:

THEOREM 6 (Non-SKYLINE CASE IV). A Case IV Path p cannot be a skyline path in $D^{\beta \cup \beta'}$ if p is dominated by the same path q in D^{β} and $D^{\beta'}$

PROOF. Because *q* can dominate *p* in both D^{β} and $D^{\beta'}$, *q* can also dominate *p* in $D^{\beta \cup \beta'}$, then *p* is not a skyline path in $D^{\beta \cup \beta'}$. \Box

THEOREM 7 (SKYLINE CASE IV). Given a Case IV Path p, suppose S_i is a set of skyline paths that can dominate p in D_i^2 , where D_i^2 denotes the $i^{th} D^2$ subspace. Then p is a skyline path in $\bigcup_{i \in [0,k]} D_i^2$ iff $\bigcap_{i \in [0,k]} S_i = \Phi$, where $k \leq {n \choose 2}$.

PROOF. This is the opposite of Theorem 6. Suppose *p* is not a skyline path, then there exists a path that can dominate *p* in D^n and every D^2 , which contradicts to $\bigcap_{i \in [0,k]} S_i = \Phi$.

Theorem 3 to 5 can derive the skyline path results from lower to higher dimensions, which satisfies our aim of "pushing down" the skyline concatenations: with several fast lower-dimensional skyline concatenations, we are able to obtain a large set of the lower dimensional results, which can be used to answer the higher dimensional queries directly. As for the results missed by the lower dimensional results, Theorem 6 and 7 help to reduce the concatenation space for the higher dimensional skyline paths.

5.3 Skyline Path Cube and Cuberization

Now we are ready to describe the structure of the skyline path cube. Firstly, a skyline cube acts like a subspace that organizes the criteria hierarchically. Apart from the subspaces, the skyline path cube also allocates the skyline paths into different cuboids of the lattice. The following definition categorizes the skyline paths according to the dimension numbers, which can be further used to allocate the paths into different levels of the cuboid:

DEFINITION 12 (*k*-**EXCLUSIVE SKYLINE PATH**). A path *p* is a *k*-exclusive skyline path if it can be derived from the D^k but not from D^{k-1} . In other words, *k* is the lowest dimension number to validate *p* as a skyline.

DEFINITION 13 (SKYLINE PATH CUBE). Given a set of skyline paths from u to v in D^n , a skyline cube $Cube_{uv}^n$ is a mapping from the k-exclusive paths to their corresponding D^k cuboids.

THEOREM 8 (**SKYLINE CUBE DISJOINT AND COMPLETENESS**). All the skyline are stored in one and only one level of the cuboids.

PROOF. Obviously, a skyline path can only appear in one level of the cuboid. As for the completeness, suppose p is a skyline path but does not exist in the cube. If p does not exist in level k, then it must be either dominated by the skyline paths in D^k , or exists in the higher levels at least as a k + 1-exclusive skyline path. If p does not exist in level k + 1, then it is at least a k + 2-exclusive skyline path. Therefore, when this procedure reaches k = n, then p is dominated by the paths in the cube, so such p does not exist. \Box

Figure 6-(b) shows an example of the skyline cube of the paths in (a). It should be noted that the skyline cube is another layer to organize the skyline paths, so the original skyline paths are still stored in a table like (a), and the cuboids only store the address of the path in the table. Firstly, the cuboid $P^{\{wc_1\}}$ has the path set $\{p_1, p_3, p_5\}$, as the projections of them on $D^{\{wc_1\}}$ are skyline paths. Secondly, we can also derive that p_1, p_2, p_3 and p_4 are skyline paths in the superspaces of their incomparable D^2 subspaces. Specifically, p_3 has three incomparable D^2 subspaces $D^{\{wc_1\}}, D^{\{c_1c_2\}}$ and $D^{\{c_1c_3\}}$. Therefore, p_3 is also the skyline path in the superspaces that contain any one of these incomparable subspaces, and it is a skyline path in all the higher spaces. As for p_5 , it is indistinct in $D^{\{wc_1\}}$ with p_1 but it is not a skyline in its superspace $D^{\{wc_1c_2\}}$ as it is dominated by p_1 . As for p_6 , it can only be found in $D^{\{wc_1c_2\}}$ and $D^{\{wc_2c_3\}}$, so it is a 3-exclusive skyline path. We call the process to construct a skyline cube from a set of paths *Cuberization* operation Cube(p). Firstly, we validate the paths from lower to higher dimensional cuboids. For the higher dimensional cubes, we also need the results from their corresponding sub-cuboids for path validation. Each time we finish a level of cuboids, we can remove these paths from P^{β} . To further reduce the index size, we propose the following skyline cube bounds to prune the impossible paths from the existing result:

LEMMA 9 (**SKYLINE PATH BOUNDARY**). Given any $D^{c_i c_j}$, its skyline paths must exist within the boundary of $[c_i.min, c_i.max] \times [c_j.min, c_j.max]$, where the values obtained from its skyline path set.

We emphasize that if a subspace path p in a $D^{c_i c_j}$ has the minimum value on c_i and on other paths have the same value with p on c_i , p must be a subspace skyline path in $D^{c_i c_j}$, and it determines the maximum value of skyline paths on c_j . Hence, we sort the paths in P on the both c_i and c_j with the increasing order. The top one path on each criterion can determine the skyline path boundary $[c_i.min, c_i.max] \times [c_j.min, c_j.max]$ if $D^{c_i c_j}$ is a incomparable subspace of them. For example in Figure 6-(a), as p_2 has $c_2.min = 7$, it determines the $c_3.max = 10$. p_1 has $c_3.min = 2$. Then we locate $c_2(p_1) = 10$ as $c_2.max$. Thus, the skyline boundary of $D^{c_2 c_3}$ is $[c_2(p_2), c_2(p_1)] \times [c_3(p_1), c_3(p_2)] = [7, 10] \times [2, 10]$.

DEFINITION 14 (**SKYLINE CUBE BOUND**). Given a set of skyline paths, its skyline cube bound is a hypercube $B = \prod_{\forall c_i \in \beta} [c_i.min, c_i.max]$ such that all the skyline path values must exist in the corresponding ranges.

To obtain a skyline cube bound of a set of skyline paths, we sort them on each criterion, and locate two skyline results of each $D^{\{c_ic_j\}}$ space: the first one on c_i that has the $c_i.min$ and $c_j.max$, and the first one c_j that has the $c_j.min$ and $c_i.max$. After that, we go through all the D^2 boundaries and take the minimum/maximum of each dimension as the final bounds. With such a bound derived from the D^2 , we can use it to prune the paths whose values are all larger than each upper bound. The time complexity of Cuberization can be loosely bounded by $O(\sum_{i=2}^{|\beta|} {|\beta| \choose i} \times n \log^{|\beta-1|} n)$, while the actual running time is much faster due to the above pruning on the higher dimensional exclusive path size.

6 SKYLINE CUBE CONCATENAITON

Different from *FHL*, whose label stores the skyline paths of a specific D^{β} , *FHL-Cube*'s label stores the skyline path cube. During the index construction and query processing, *FHL-Cube* needs to concatenate two skyline path cubes in all non-empty subspaces instead of from only one D^{β} . In the following, we present how to concatenate the skyline cubes efficiently in the single (Section 6.1) and multiple (Section 6.2) hop scenarios, and how to utilize them during query processing (Section 6.3).

6.1 Single Hop Cube Concatenation

DEFINITION 15 (**SKYLINE PATH CUBE CONCATENATION**). Given two skyline path cubes $Cube_{uh}^n$ and $Cube_{hv}^n$, the concatenation result $Cube_{uhv}^n = Cube_{uh}^n \oplus Cube_{hv}^n = \{Cube_{uh}^\beta \oplus Cube_{hv}^\beta, |\forall D^\beta \subseteq D^n\}$ contains all the skyline paths from u to v via h, and these skyline paths are organized in a skyline path cube $Cube_{uhv}^n$. The concatenation operation includes two phases: 1) Add up the path values to generate a concatenated path set P_s ; 2) Find the skyline paths from P_s for computing skyline cubes. As we can not guarantee that all concatenated paths are still skyline paths [36, 37], it will increase the complexity of computing skyline cubes. For example, Table 1a and 1b are two D^4 skyline path sets P_{uh} and P_{hv} . The $D^{\{c_1c_3\}}$ skyline paths in P_{uh} are p_1 and p_3 , while P_{hv} 's are p'_2 and p'_3 . When concatenate $Cube_{uh}^{c_1c_3} \oplus Cube_{hv}^{c_1c_3}$, we can generate four candidates $p_1^2 = (8, 10, 18, 5), p_1^3 = (5, 9, 19, 6), p_3^2 = (7, 7, 19, 6), p_3^3 =$ $(8, 8, 18, 7). p_1^3$ and p_3^3 are dominated by p_3^2 in $D^{\{c_1c_3\}}$. Therefore, the final result of $Cube_{uhv}^{\{c_1c_3\}}$ is $P_{uh}^{\{c_1c_3\}} \oplus P_{hv}^{\{c_1c_3\}} = \{p_1^2, p_3^2\}$.

Table 1: Skyline Path Concatenation of P_{uw} and P_{wv}

(a) Skyline Path Set P _{uh}						(b) Skyline Path Set P _{hv}					
P_{uh}	w	<i>c</i> ₁	c_2	<i>c</i> ₃	[P_{hv}	w	c_1	c_2	<i>c</i> ₃	
<i>p</i> ₁	2	4	10	2		p'_1	6	6	8	3	
p ₂	3	7	7	10	Ĩ	p'_2	2	4	10	3	
<i>p</i> ₃	5	3	9	3		p'_3	3	5	9	4	

We improve the concatenation efficiency from two aspects: 1) *Block Maintenance* when concatenating two underlying skyline paths; 2) *Skyline Path Derivation* when identifying all subspace skyline paths in each *Cube*. In the following, we first introduce 1) for reducing the size of the concatenated paths, which can further reduce the computation time of *Cubes* in 2).

Block Maintenance. Given two sets of *m* and *n* skyline paths, the concatenation paths number *mn* could be large and deteroiraets the cube concatenation efficiency. To reduce *mn* to a smaller value, we first introduce the *path entropy* [14].

DEFINITION 16 (**PATH ENTROPY** ϵ). Given a n-D path $p \in P_s$, its entropy $\epsilon(p) = lnw(p) + \sum_{i=1}^{n-1} ln c_i(p)$. If $\epsilon(p)$ is small, then p has higher probability to dominate paths in P_s .

Specifically, we maintain a *Block* of *k* smallest entropy concatenated paths. For each $p_i \in P_s$, we can discard it if dominated by any path in *Block*. Otherwise, we compute its entropy $\epsilon(p_i)$ and replace p_j in *Block* if $\epsilon(p_j) > \epsilon(p_i)$. In addition, the *Block* maintenance can be further optimized in the *tree node contraction* phase where the cubes are concatenated when we compute skyline shortcuts between the neighbour pairs of each vertex *w*. If an edge e = (u, v)exists between *w*'s two neighbours *u* and *v*, then it already includes a skyline path set $P_{u,v}$, and we need to merge the concatenated paths $P_{u,w,v}$ with $P_{u,v}$. As $|P_{u,w,v}|$ can be much larger than $|P_{u,v}|$, we use the *k* smallest entropy paths in $P_{u,v}$ as the initial *Block*. In practice, this method is effective especially in the boundary tree.

Cuberization with Skyline Path Derivation. We propose to break down the expensive high-dimensional cube concatenation into several cheap low-dimensional ones using the previously discussed skyline path derivation and *Cube* prunings.

Given two skyline path sets P_1 and P_2 , we first add their path values to obtain an all-pair concatenated path set P_s . Then we sort the paths in P_s in ascending order on each criterion. As shown in Figure 7-(a), $\forall p_i \in P_s$, the paths ranking before p_i can dominate p_i . On the contrary, the paths after p_i are dominated by p_i . We denote the set of paths dominating p_i on a subspace D^β as $\mathcal{DG}^\beta(p_i)$. Then, we pair p_i to the correct $Cube^\beta$ by checking whether there is a common



Figure 7: Sorting List and Pruning Area Examples

path p'_i ranking before p_i on each criterion in D^{β} . For example, if $\mathcal{D}\mathcal{G}^w(p_i) \cap \mathcal{D}\mathcal{G}^{c_1}(p_i) = p'_1, \mathcal{D}\mathcal{G}^w(p_i) \cap \mathcal{D}\mathcal{G}^{c_3}(p_i) = p'_2$, we can determine that D^{wc_1} and D^{wc_3} are two dominated subspace $D^{wc_1}_{<}(p_1)$ and $D^{wc_3}_{<}(p_1)$. As $\mathcal{D}\mathcal{G}^w(p_i) \cap \mathcal{D}\mathcal{G}^{c_1}(p_i) \cap \mathcal{D}\mathcal{G}^{c_3}(p_i) = \phi$, no path can dominate p_i in $D^{wc_1c_3}$. Thus, $wc_1c_3(p_1)$ is a 3exclusive skyline path and recorded in a *Cube* wc_1c_3

Next, we find these common paths from lower to higher dimensions. Firstly, we merge all the $\mathcal{DG}^1(p_i)$ and form a new list $\mathcal{L}(p_i)$ as shown in Figure 7-(c). Note that the paths p'_i in any $\mathcal{DG}^1(p_i)$ can be ordered by the record location on the concatenated path set P_s . The order range of p'_i is in [0, mn - 1]. Therefore, we can create a hash-table based on the paths' subscripts and the locations of P_s . It reduces the space complexity and avoids redundant access on \mathcal{L} . Specifically, we scan $\mathcal{L}(p_i)$ from the start. When we visit each path p'_i , we mark on the *i*th location of P_s and record the current \mathcal{DG}^1 . During the scanning, once we visit p'_i , the *i*th location on P_s will be marked again and another $\mathcal{DG}^{1}(p_{i})$ is recorded. After finishing the scan, the locations of P_s marked more than twice will be extracted and we can also get their $\mathcal{DG}^1(p_i)$. For example in Figure 7-(c), when we scan $\mathcal{DG}^{c_1}(p_i)$ and encounter p'_1 , we mark 1^{st} location on P_s and record c_1 at the same time. As we have scanned $\mathcal{DG}^w(p_i)$ already, the 1^{st} location now is marked twice and recorded wc_1 . We can determine D^{wc_1} as $D^{wc_1}_{\prec}(p_i)$. In this way, we can find all the $D^2_{\prec}(p_i)$ and the remaining D^2 s are $Cube^2(p_i)$ s.

The *Cubes* of p_i on the upper level of $Cube^2(p_i)$ can be directly obtained by the intersection operation on determined D_{\leq}^2 and the corresponding subspaces of $Cube^2(p_i)$. However, we still have room to improve the computing efficiency owing to the theorem below:

THEOREM 10 (**SKYLINE PATH CUBE PRUNING**). Given a n-D path p_1 , once we determine a dominated subspace $D^{\beta}_{<}$, $D^{\beta} \subseteq D^n$, we can prune D^{β} and all its subspaces for computing Cubes of p_1 .

PROOF. p_1 exists in $Cube_{uv}^{\beta}$ only if $\beta(p_1)$ is a subspace skyline path. Then D^{β} can be pruned. For the subspaces, the only case that p_1 is a skyline path in the subspace of D^{β} is when $\beta(p_1)$ has at least one indistinct subspace $S \subset \beta$ and one dominated subspaces. Therefore, it is a *Case III* path and must have another path $S(p'_1)$ including the same path value on S with $S(p_1)$. As p_1 is dominated in the superspace of S, there always exists a path p'_1 , which is better than p_1 . Therefore, pruning p_1 in all the subspaces of D^{β} can not affect the correctness of the query result. As shown in Figure 7-(b), we can prune a part of the subspaces by computing *Cubes* of any *n*-D $p_i \in P_s$ based on the results of $D^2_{\prec}(p_i)$ and $Cube^2(p_i)$. Firstly, we can prune all superspaces of each determined $Cube^2(p_i)$ (green shaded area) by referring to Definition 12. Suppose we compute $T Cube^2$ of p_i , each $Cube^2(p_i)$ can prune $\sum_{i=0}^{n-2} {n-2 \choose i}$ subspaces of p_i . Therefore, we can totally prune $T \cdot \sum_{i=0}^{n-2} {n-2 \choose i} - \sum_{i=1}^{k} {k \choose i}$ subspaces, where $\sum_{i=1}^{k} {k \choose i}$ is the number of the common subspaces, and the value of $k \in [T + 1, 2T]$.

Note that our algorithm can discover all the $DG^{max(\beta)}(p_i)$ after scanning on $\mathcal{L}(p_i)$, where $max(\beta)$ refers to the maximum dominated subspace of p_i . Then we prune all the subspaces of $max(\beta)$, as shown in the blue shaded area. For example in Figure 7-(c), p'_4 is marked the most times, the corresponding criteria form the $DG^{wc_2c_4}(p_i)$. Thus, all subspaces of wc_2c_4 can be pruned.

In this step, it will cover one or more pruned $D^2_{\prec}(p_i)$. Suppose the total number of $D^2_{\prec}(p_i)$ is $M < \binom{n}{2} - T$, and the number of pruned ones is R, which results in $R \cdot \sum_{i=1}^{|max(\beta)|} - \sum_{i=1}^{k} \binom{k}{i}$ pruned subspaces, where $k \in [|max(\beta)| + R - 1, |max(\beta)| \cdot R]$. Consider the *Case IV path*, p_i can be a subspace skyline path in a set of subspaces S, including all the remaining subspace except for the D^2 ones. Note that all subspaces in S are the skyline subspaces of p_i . However, we only need to find the lowest dimensional subspaces in S as the *Cubes* of p_i in terms of Definition 13.

6.2 Multiple Cube Concatenation Pruning

During the label construction and query answering, we need the skyline concatenation result $Cube_{uh_iv}^{\beta}$ on several hops with $h_i \in H = L(u) \cap L(v)$. The straightforward way is computing the cubes of these h_i , merging the results, and constructing the new cube $Cube_{uv}^{\beta}$. However, because the cube concatenation has a high complexity, and a pair of vertices could have hundreds of such hops, which further creates a large number of paths to check, it is very slow (several seconds) to finish one concatenation. What is worse, the current hop-first concatenation paradigm, which finishes all cube concatenations as a whole, makes it hard to prune useless concatenations beforehand. In query processing, we change the concatenation order from the hop-first perspective to the subspace-first perspective according to the number of constraints and propose a rectangle-based subspace pruning technique to reduce the number of hops and the corresponding cuboids.

Firstly, for each $Cube^2$, we find the bounds of each hop h_i by concatenating their first and last skyline paths. In this way, we obtain a virtual rectangle r_i for each hop such that their concatenation results all falling into it. Then for any two rectangles r_i and r_j , if r_i 's lower-left point is dominated by r_j 's upper-right, we can safely prune all the points in r_i . In other words, we can avoid concatenating h_i in this subspace. Therefore, a linear scan of the rectangles is enough to obtain a candidate hop set for each D^2 . For example in Figure 8-(a), h_5 and h_6 are dominated by h_1 , h_2 , and h_3 , so we can avoid concatenating h_5 and h_6 safely. As for h_4 , although the right part of it is dominated, we still have to keep it as the left part could still be a skyline result. After that, for the higher dimensional cuboids, we only need to take the union results of its corresponding hop candidates, which is proved in the following theorem:



Figure 8: Rectangle-based Hop Pruning

THEOREM 11. A hop h is a candidate in D^{β} if it exists in one of its subspaces.

PROOF. Firstly, it is trivial to prove that the union of the subspace's rectangles is equivalent to taking the skyline cube's bounds. Then the rectangles that are out of a cube's bounds are dominated in this subspace. Therefore, those hops associated with the dominated rectangles can be pruned safely. Because we did not prune any correct hop, the remaining ones are correct candidates.

By organizing the candidate hops in different subspaces, we can obtain the candidate hop cube \mathcal{H} . We first obtain the candidate hops for each D^2 subspaces. After that, the candidates propagate upward to the candidate set of their superspaces. After all the superspaces have obtained their candidates, we organize them into a hop cube to guide the actual hop concatenations in each subspace, as shown in Figure 8-(c). Figure 8-(b) illustrates an example of deriving the D^3 candidates from its corresponding D^2 candidates. The hops that are covered by the new cube are also potential results, so we only need to take the union of the three D^2 candidate sets (blue, green, and orange regions). The complexity of the pruning phase is linear to $O(|H| \log |H|)$, which is spent on the rectangle sorting. As |H|is much smaller than the skyline path number |P|, its complexity is dominated by the cube concatenation. Therefore, the pruning phase reduces the concatenation number at a very small cost.

6.3 Flexible MCSP Query Answering

The following theorem first proves the information we need to answer a flexible *MCSP* query correctly:

THEOREM 12. Given a MCSP query $Q(s, t, C^{\beta})$, its result could only exist in the Cube^{β}

PROOF. Because the *MCSP* results can only be one of the skyline paths as proved in Theorem 1, we only need to guarantee that the paths not in $Cube^{\beta}$ are dominated in D^{β} , which is ensured by the definition of the skyline cube.

Therefore, we only need to obtain the $Cube_{st}^{\beta}$ to answer $Q(s, t, C^{\beta})$. To further reduce the computational cost, we can apply the constraints such that only the paths satisfying the constraints are needed. In this way, only a smaller cube $Cube_{st}^{\beta}(C)$ is needed. In the following, we describe how to obtain the $Cube_{st}^{\beta}(C)$ when s and t are in the same or different partitions.

Inner Partition Query. When *s* and *t* are in the same partition, we only need to use one inner skyline cube tree. Firstly, we find the *LCA* X_u of X_s and X_t , then all the vertices $\{h_i\}$ in X_u are

the hops from s to t. Secondly, we apply multiple cube concatenations based on these hops $\{Cube_{sh_i}^{\beta} \oplus Cube_{h_it}^{\beta}\}$. To further reduce the concatenated hop number, we replace the upper-bounds of dimension with the constraints in C. In this way, smaller rectangles are obtained while more hops are pruned. Besides, during the hop concatenation, we also apply the constraints on the cubes to be concatenated. In other words, $Cube_{sh_i}^{\beta} \oplus Cube_{h_it}^{\beta}$ is replaced with $Cube_{sh_i}^{\beta}(C) \oplus Cube_{h_it}^{\beta}(C)$. Finally, as MCSP only needs one constraint-satisfying path with the smallest weight, early termination can be applied to further improve the efficiency. Specifically, we only concatenate the paths that satisfying the constraints in the weight-increasing order. Hence, when the first concatenated path satisfies the constraints appears, we can stop concatenating and return it as the final result.

External Partition Query. Suppose *s* and *t* are in different partitions, and their corresponding boundary sets are denoted as B_s and B_t . The inter partition query can be decomposed into three sub-queries: i) from *s* to B_s , from ii) B_s to B_t , and iii) from B_t to *t*. The results of the i) and iii) can be obtained directly without concatenation because the boundaries must be the ancestors of the inner vertices. The results of ii) can be obtained as a query on the boundary tree. As for the concatenation results of i) and ii), we can view B_1 as the hops from *s* to each $b_2 \in B_2$. Therefore, multi-hop pruning can also be applied here. After that, we get the cubes from *s* to B_2 , and can also view each $b_2 \in B_2$ as hops from *s* to *t*. Besides, the aforementioned cube pruning using the constraints can also be applied here to further reduce the concatenation computation.

6.4 Extension For Directed Graph

We denote the directed version as FHL-Cube+.

Indexing. For both *Inner Tree* and *Boundary Tree*, *FHL-Cube*+ is similar to *FHL-Cube* with two differences: 1) The neighbours of any vertex v are classified into *in*-neighbours $N_i(v)$ and *out*neighbours $N_o(v)$. During contraction, we iteratively concatenate one $u \in N_i(v)$ and one $w \in N_o(v)$ to form a directed pair (u, w). 2) Instead of saving skyline cubes between v and v's neighbours as labels in each node X(v), we store $Cube_{u,v}$ as *in*-label and $Cube_{v,w}$ as *out*-label separately to distinguish their directions towards v. Hence, when we assign labels between v to all its ancestors in X(v), we consider the vertices in $N_i(v)$ as *in*-hops and the vertices in $N_o(v)$ as *out*-hops to compute the skyline cubes from its ancestors to v as *in*-labels and v to its ancestors as *out*-labels separately.

Query Processing. Given a query $q(s, t, C^{\beta})$, suppose s and t are in the same partition, we first compute the $LCAX_u$ of X_s and X_t . In X_u , we find the vertices, which have the *out*-label with s and *in*-label with t, as the hops. After that, we compute the cubes from s to the selected vertices and from the vertices to t under the constraints C^{β} . We can use the same way to obtain the results on cube trees and boundary tree if s and t are in different partitions.

7 EXPERIMENT

7.1 Experimental Settings

We implement all the algorithms in C++ with full optimization. The experiments are conducted in a 64-bit Ubuntu 18.04.3 LTS with two 8-cores Intel Xeon CPU E5-2690 2.9GHz and 186GB RAM.

Datasets. We conduct experiments on the following four realworld networks [1]: 1) *NY* is a dense grid-like urban area with 264,246 vertices, 733,846 edge, and several partitions connected by bridges; 2) *BAY* has a shape of doughnut around the *San Francisco Bay* with 321,270 vertices, 800,172 edges; 3) *COL* has an uneven vertex distribution (dense around Denver but sparse elsewhere) with 435,666 vertices and 1,057,066 edges; 4) *FLA* is a large network Florida with 1,070,376 vertices and 2,712,798 edges. We use the road length as the weight w and travel time t as the first cost. And three more positive correlation costs are generated randomly. We also obtain three additional real-world criteria for *NY-REAL*: 1) *Toll Charge* from NY City Council, 2) *Traffic Light* from OpenStreetMap, and 3) *Attractiveness* from geo-tagged *Flickr* photos [41].

Algorithms. We compare the proposed *FHL-Cube* with the existing exact *MCSP* solutions: 1) *FHL* [37]: index for full space; 2) *FHL-Multi* [37]: index for each subspace; 3) *Sky-Dij* [18]; 4) *eKSP* [50]; 5) *CSP-CH* [52]. We do not compare with *COLA* [56] here since its query results are approximate. Among them, *Sky-Dij* and *eKSP* are index-free approaches and the index size of *CSP-CH* is out of memory when the dimension is larger than three.

Query Set. For each dataset, we randomly generate three sets of OD pairs Q_1 to Q_3 and each has 1000 queries. Specifically, we first estimate the diameter d_{max} of each network [42]. Then each Q_i represents a category of OD pairs falling into the distance range $[d_{max}/2^{4-i}, d_{max}/2^{3-i}]$. Then we assign the query constraints C to these OD pairs. Firstly, we compute the cost range $[C_{min}, C_{max}]$ for each OD. This is because if $C < C_{min}$, the optimal result can not be found then the query is invalid; and if $C > C_{max}$, the optimal result is the path with the shortest distance. Furthermore, to test the constraints for each OD pair: $C = r \times C_{max} + (1 - r) \times C_{min}$, with r = 0.1, 0.3, 0.5, 0.7 and 0.9. When r is small, C is closer to the minimum cost; and when r is big, C is closer to the maximum cost.

7.2 Experiment Results

Index Size and Construction Time. The results are shown in Figure 9. Since Sky-Dij and eKSP are index-free, they are not compared here. As for the CSP-CH, its construction time soars up to five orders of magnitude from 2-D to 3-D, which is caused by the explosion of the skylines on the entire graph as the dimension increases. Consequently, it fails to construct in the 4-D and 5-D graphs, so we do not show its results. FHL-Multi takes the longest time with the largest index since it constructs labels for each subspace. FHL constructs the index only for the full space, so it has the smallest index size. However, its construction time is still very long. For FHL-based algorithms, the most time-consuming part is on boundary tree construction, as the boundary graph are extremely dense and most edges stores skylines. Compared with FHL and FHL-Multi on indexing, FHL-Cube has the significant reduction on time cost and performs the best (FHL spends 15631 seconds on 5dimensional indexing of NY, while FHL-Cube takes 7861 seconds). Its efficiency mainly embodies in two aspects: first, our principles of boundary tree construction reduce the tree height obviously, and then decrease the computation from a tree node to its ancestors corresponding to the vertices in other partitions; second, in each contraction of tree construction, Block Maintenance reduces many

concatenated paths before the cuberization operation. In addition, its index size is much smaller than the *FHL-Multi*, but has a larger index than *FHL*. It is because our skyline cube is essentially another layer of path management index, so it may store the address of the same path multiple times at different cuboids. Nevertheless, it is still faster to construct even with larger index size.



Figure 9: Index Size and Construction Time

Constraint Number. As shown in Figure 10, the query time of eKSP, Sky-Dij and CSP-CH are orders of magnitude higher than that of the FHLs'. This is because eKSP has to enumerate a huge amount of paths before one can satisfy the constraints, while the graph-search methods have accumulated a huge amount of intermediate skyline results and have to traverse the same edge multiple times. The performance of FHL with different constraint numbers is almost in the same order of magnitude because they use the same whole-space index. FHL-Multi is always faster than FHL in lower dimensions since it has an independent index in each subspace. The performance of all comparable algorithms deteriorate as the dimension grows. When we reach D^5 , they present the same performance because they use exactly the same 5-D skyline index. FHL-Cube has the similar performance with FHL-Multi because the index size they use for concatenation is the same. However, it is faster than FHL in the other dimensions, because FHL answers the queries by the full-dimensional index, which results in more concatenated paths. Hence it will spend more time on identifying an optimal skyline path under the constraints. On the other hand, FHL-Cube can locate the subspace skyline index according to the constrained criteria. It also provides more powerful hop pruning on the corresponding subspace and produces fewer concatenated results.

Query Distance and Constraint Ratio Since the query processing efficiency of *eKSP*, *Sky-Dij* and *CSP-CH* are far behind the *FHLs* as shown in Figure 10, we do not include their performance in this section. As shown in Figure 11, the query time increases slightly as the query distance becomes longer. This is because that the longer path tends to pass through more partitions, which could cause more path concatenation. Nevertheless, the query performance is stable regardless of the distance changes. With the increase of constraint ratio, the query performance of the *FHLs* also keeps stable while other methods are very sensitive to the constraint ratio [36, 67]. Moreover, all of them perform the best when the constraint ratio is set to 0.1. In the case, we can prune more hops, because we can give a smaller concatenation range ahead of time. In summary, our method is robust to different query distances and constraint ratios.



Figure 10: Comparison with MCSP Baselines ($r = 0.5, Q_3$)



Figure 11: Query Time (3-Dimension)



Figure 12: Flexible MCSP Query Time (Q_3 , r = 0.5)

Flexible MCSP. To test the performance of the flexible *MCSP* query, we generate the criteria combinations randomly. The number of constraints is set from 1 to 4 in Q_3 with r = 0.5, and the results are shown in Figure 12. *FHL-Multi* and *FHL-Cube* have the similar performance due to the similar index sizes used on any combinations. The performance of *FHL-Cube* is better than *FHL*, and the advantage is more obvious in larger graph like *COL*, because *FHL* always uses the full-dimensional index and produces more concatenated results. Conversely, *FHL-Cube* can find a smaller index size due to the hierarchical structure of *Cube*, and provide a smaller bound to prune calculative hops as much as possible.

Real Road Network. We test both directed and undirected on *NY Real*, with directed verions are labeled with "+". Moreover, each pair of adjacent vertices has two-way edges with the same attributes. As shown in Figure 13, the directed ones take double the construction and query time of the undirected ones with double index size. Both directed and undirected versions of *FHL-Cube* have the similar performance with *FHL-Multi* but better than *FHL*, because the random criterion *Attractiveness* produces a large number of skylines with *distance*. The number of skyline paths in the 5-D graph is far greater than the 2 to 4-D graphs'. Therefore, *FHL* has to consider several times more skylines than both *FHL-Cube* and *FHL-Multi*.



Figure 13: NY Real Criteria and Directed Evaluation

Remarks. Overall, our *FHL-Cube* is more efficient and practical to answer flexible *MCSP* queries than the baselines. Although the query performance of *FHL-Cube* is similar with *FHL-Multi*, its index size is much smaller. On the other hand, *FHL-Cube* achieves better query performance than *FHL* on both flexible and fixed *MCSP* query processing. As a result, even though its index size is slightly larger than that of *FHL*, it takes shorter time to construct thanks to the efficient cube concatenation and optimized index structure. Given these points, *FHL-Cube* is more suited for flexible *MCSP* querying.

8 RELATED WORK

8.1 Shortest Path and Skyline Path

Sky-Dijk is the first skyline path approach that searches the space in similar way as *Dijkstra*'s while one vertex could be traversed several times for multiple skyline results. [27] incorporated with *landmark* [26] prunes the search space by estimating the lower bound and proposing the dominance relations. The skyline operator is utilized in [15] to find a set of skyline destinations. *Keyword-Aware* routing [24, 35, 45, 57] involves skylines based on stop number, satisfied keywords, and travel time. [60] further applies user preference function on the skyline paths. As for the path indexes, *2-hop labeling* [4, 7, 34, 43, 63, 65, 66] is the state-of-the-art.

8.2 Constrained Shortest Path

Single Constraint Shortest Path. We discuss CSP algorithms from the perspective of exact/approximate and index-free/index. 1) Exact Index-Free CSP: CSP is first studied in [21] with dynamic programming but it can hardly scale. CSP can also be solve with Sky-Dijk by pruning over the constraint, and k-Path enumeration [11, 39, 50, 61] by testing the paths one by one. [40] accelerate the Sky-Dijk search with GPU. 2) Approximate Index-Free CSP: The approximation in CSP problem relaxes the length of final path to be $(1 + \alpha)$ times of the optimal path at maximum through *linear* programming [18, 38] and Lagrange Relaxation [5, 17, 23]. However, they are even slower than the k-Path-based exact approaches [28, 39]. *CP-CSP* distributes the approximation power $\sqrt[n]{\alpha}$ to the edges based on Sky-Dijk, but its performance is only slightly better since its approximation ratio would decrease to 1. 3) Exact CSP Index: CH [13] is the only shortest path index which has been extended to solve CSP [52] with its shortcuts being skyline paths. Nevertheless, it suffers from long index construction time and large index size because both the index construction is based

on *Sky-Dijk*, and its query performance is low because it is essentially *Sky-Dijk*-based. **4) Approximate CSP Index**: *COLA* [56] is the only approximate index-based method. It partitions the graph into regions and precomputes approximate skyline paths between regions in *Sky-Dijk* fashion. It views α as a budget and concentrates the pruning power on the important vertices.

Multi-Constraint Shortest Path. Most existing *MCSP* algorithms have limited scalability to real-life networks with high complexity. For instance, [20] is of complexity $O(|V|^5 c_{max} \log(|V|c_{max}))$. [25] computes the approximate *MCSP* heuristically but it only scales to small graph (200 vertices) and takes several seconds to compute on graphs with 100 vertices [51]. *Pareto-SHARC* [9] . [25] extends *SHARC*, which was already proven worse than *CH* and no longer used nowadays. Since the skyline path is restricted to the predefined "preference function" [27, 59], they does not apply to *MCSP* problem. The experiments about skyline-based and *k*-Path-based methods are reported in [67]. *FHL* [37] is the only label-based index and it has been analyzed thoroughly.

8.3 High Dimensional Skyline Cube

The existing high dimensional skyline methods (*Skyey* [48], *Stellar* [47], and *Orion* [49]) all aim to find the skyline results from different subspaces from the high dimensional space (Skyline Cube Operation). However, our flexibile *MCSP* requires to derive the high dimensional results from the lower ones, and our path concatenation requires concatenation two sets of skylines, which are not supported by these methods, so they are unrelated to our problem.

9 CONCLUSION

In this paper, we studied the flexible MCSP problem, which is the most generalized and practical multi-objective/constraint routing problem, and proposed FHL-Cube index to handle it efficiently. By establishing the skyline path relation thoeries between the lower and higher dimensions, we introduced cube to organzie the skyline paths in different subspaces, derived higher dimensional skyline path from lower ones, and improved the single- / multi-hop skyline cube concatenation for both index construction and query processing. We reduce the FHL-Cube construction time by optimizing the structure and concatenation strategies. Extensive evaluations on real-life road networks show the priority of FHL-Cube compared with the state-of-the-art approaches in terms of the construction time, index size, query processing efficiency and flexibility. Finally, to cope with the dynamic criteria like travel time, we could either resort to the speed profiles [62] and create indexes for each time slot, or resort to index maintenance techniques [63, 65, 66] to update the skyline path labels accordingly.

ACKNOWLEDGMENT

This work was partially supported by the Australian Research Council under Grant No. DP200103650 and LP180100018, Hong Kong Research Grants Council (grant# 16202722), and was partially conducted in the JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust.

REFERENCES

- [1] [n.d.]. 9th DIMACS Implementation Challenge Shortest Paths. http://users. diag.uniroma1.it/challenge9/download.shtml.
- [2] [n.d.]. Complete version of this paper. https://www.dropbox.com/s/ 3nr2jnch7ok0l4b/documentFull.pdf?dl=0.
- [3] Jonathan D Adler and Pitu B Mirchandani. 2014. Online routing and battery reservations for electric vehicles with swappable batteries. *Transportation Research Part B: Methodological* 70 (2014), 285–302.
- [4] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In Proceedings of the 2013 ACM SIGMOD international conference on Management of data. ACM, 349–360.
- [5] W Matthew Carlyle, Johannes O Royset, and R Kevin Wood. 2008. Lagrangian relaxation and enumeration for solving constrained shortest-path problems. *Networks: An International Journal* 52, 4 (2008), 256–270.
- [6] Lijun Chang, Jeffrey Xu Yu, Lu Qin, Hong Cheng, and Miao Qiao. 2012. The exact distance to destination in undirected world. VLDB Journal 21, 6 (2012), 869–888.
- [7] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and distance queries via 2-hop labels. SIAM J. Comput. 32, 5 (2003), 1338–1355.
- [8] Hans De Neve and Piet Van Mieghem. 1998. A multiple quality of service routing algorithm for PNNI. In 1998 IEEE ATM Workshop Proceedings. 324–328.
- [9] Daniel Delling and Dorothea Wagner. 2009. Pareto paths with SHARC. In International Symposium on Experimental Algorithms. Springer, 125–136.
- [10] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. Numerische mathematik 1, 1 (1959), 269–271.
- [11] Jun Gao, Huida Qiu, Xiao Jiang, Tengjiao Wang, and Dongqing Yang. 2010. Fast top-k simple shortest paths discovery in graphs. In Proceedings of the 19th ACM international conference on Information and knowledge management. 509–518.
- [12] Michael R Garey and David S Johnson. 1979. Computers and intractability. Vol. 174. freeman San Francisco.
- [13] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *International Workshop on Experimental and Efficient Algorithms*. Springer, 319–333.
- [14] Parke Godfrey, Ryan Shipley, Jarek Gryz, et al. 2005. Maximal vector computation in large data sets. In *Proceedings of the VLDB Endowment*, Vol. 5. Citeseer, 229– 240.
- [15] Qixu Gong, Huiping Cao, and Parth Nagarkar. 2019. Skyline queries constrained by multi-cost transportation networks. In 2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 926–937.
- [16] Jason Hahn. [n.d.]. New York City to Google: Reduce the number of left turns in Maps navigation directions. https://www.foxnews.com/auto/new-york-city-togoogle-reduce-the-number-of-left-turns-in-maps-navigation-directions.
- [17] Gabriel Y Handler and Israel Zang. 1980. A dual algorithm for the constrained shortest path problem. *Networks* 10, 4 (1980), 293–309.
- [18] Pierre Hansen. 1980. Bicriterion path problems. In Multiple criteria decision making theory and application. Springer, 109–127.
- [19] Refael Hassin. 1992. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations research* 17, 1 (1992), 36–42.
- [20] Jeffrey M Jaffe. 1984. Algorithms for finding paths with multiple constraints. Networks 14, 1 (1984), 95–116.
- [21] Hans C Joksch. 1966. The shortest route problem with constraints. Journal of Mathematical analysis and applications 14, 2 (1966), 191–197.
- [22] Nicolas Jozefowiez, Frédéric Semet, and El-Ghazali Talbi. 2008. Multi-objective vehicle routing problems. *European journal of operational research* 189, 2 (2008), 293–309.
- [23] Alpar Juttner, Balazs Szviatovski, Ildikó Mécs, and Zsolt Rajkó. 2001. Lagrange relaxation based method for the QoS routing problem. In Proceedings IEEE IN-FOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society, Vol. 2. IEEE, 859– 868.
- [24] Arzoo Katiyar, Arnab Bhattacharya, and Shubhadip Mitra. 2014. Efficient and effective route planning in road networks with probabilistic data using skyline paths. In Proceedings of the 1st IKDD Conference on Data Sciences. 1–10.
- [25] Turgay Korkmaz and Marwan Krunz. 2001. Multi-constrained optimal path selection. In *INFOCOM*, Vol. 2. 834–843.
- [26] Hans-Peter Kriegel, Peer Kröger, Peter Kunath, Matthias Renz, and Tim Schmidt. 2007. Proximity queries in large traffic networks. In ACM GIS. 1–8.
- [27] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. 2010. Route skyline queries: A multi-preference path planning approach. In *IEEE International Conference on Data Engineering (ICDE)*. IEEE, 261–272.
- [28] Fernando Kuipers, Ariel Orda, Danny Raz, and Piet Van Mieghem. 2006. A comparison of exact and ε-approximation algorithms for constrained routing. In NETWORKING. Springer, 197–208.
- [29] Lei Li, Wen Hua, Xingzhong Du, and Xiaofang Zhou. 2017. Minimal on-road time route scheduling on time-dependent graphs. PVLDB 10, 11 (2017), 1274–1285.

- [30] Lei Li, Sibo Wang, and Xiaofang Zhou. 2019. Time-Dependent Hop Labeling on Road Network. In 2019 IEEE 35th International Conference on Data Engineering (ICDE). 902–913.
- [31] Lei Li, Sibo Wang, and Xiaofang Zhou. 2020. Fastest Path Query Answering using Time-Dependent Hop-Labeling in Road Network. *IEEE Transactions on Knowledge and Data Engineering* (2020).
- [32] Lei Li, Mengxuan Zhang, Wen Hua, and Xiaofang Zhou. 2020. Fast Query Decomposition for Batch Shortest Path Processing in Road Networks. In 2020 IEEE 36th International Conference on Data Engineering (ICDE).
- [33] Lei Li, Kai Zheng, Sibo Wang, Wen Hua, and Xiaofang Zhou. 2018. Go slow to go fast: minimal on-road time route scheduling with parking facilities using historical trajectory. *The VLDB Journal* 27, 3 (2018), 321–345.
- [34] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Scaling distance labeling on small-world networks. In Proceedings of the 2019 ACM SIGMOD international conference on Management of data. 1060–1077.
- [35] Yujiao Li, Weidong Yang, Wu Dan, and Zhipeng Xie. 2015. Keyword-aware dominant route search for various user preferences. In *International Conference* on Database Systems for Advanced Applications. Springer, 207–222.
- [36] Ziyi Liu, Lei Li, Mengxuan Zhang, Wen Hua, Pingfu Chao, and Xiaofang Zhou. 2021. Efficient Constrained Shortest Path Query Answering with Forest Hop Labeling. In 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 1763–1774.
- [37] Ziyi Liu, Lei Li, Mengxuan Zhang, Wen Hua, and Xiaofang Zhou. 2022. Multi-Constraint Shortest Path using Forest Hop Labeling. *The VLDB Journal* (2022).
- [38] Dean H Lorenz and Danny Raz. 2001. A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters* 28, 5 (2001), 213–219.
- [39] Leonardo Lozano and Andrés L. Medaglia. 2013. On an exact method for the constrained shortest path problem. *Computers Operations Research* 40, 1 (2013), 378 - 384.
- [40] Shengliang Lu, Bingsheng He, Yuchen Li, and Hao Fu. 2020. Accelerating exact constrained shortest paths on GPUs. (2020).
- [41] Ying Lu and Cyrus Shahabi. 2015. An arc orienteering algorithm to find the most scenic path on a large-scale road network. In Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. 1–10.
- [42] Ulrich Meyer and Peter Sanders. 2003. Δ-stepping: a parallelizable shortest path algorithm. Journal of Algorithms 49, 1 (2003), 114–152.
- [43] Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. 2018. When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In Proceedings of the 2018 ACM SIGMOD international conference on Management of data. 709–724.
- [44] Dian Ouyang, Long Yuan, Fan Zhang, Lu Qin, and Xuemin Lin. 2018. Towards efficient path skyline computation in bicriteria networks. In International Conference on Database Systems for Advanced Applications. Springer, 239–254.
- [45] Shiladitya Pande, Sayan Ranu, and Arnab Bhattacharya. 2017. SkyGraph: Retrieving regions of interest using skyline subgraph queries. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1382–1393.
- [46] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2003. An optimal and progressive algorithm for skyline queries. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data. 467–478.
- [47] Jian Pei, Ada Wai-Chee Fu, Xuemin Lin, and Haixun Wang. 2007. Computing compressed multidimensional skyline cubes efficiently. In 2007 IEEE 23rd International Conference on Data Engineering. IEEE, 96–105.
- [48] Jian Pei, Wen Jin, Martin Ester, and Yufei Tao. 2005. Catching the best views of skyline: A semantic approach based on decisive subspaces. *Space (X, Y)* 100, 4 (2005), 1.
- [49] Chedy Raïssi, Jian Pei, and Thomas Kister. 2010. Computing closed skycubes. Proceedings of the VLDB Endowment 3, 1-2 (2010), 838–847.
- [50] Antonio Šedeño-Noda and Sergio Alonso-Rodríguez. 2015. An enhanced K-SP algorithm with pruning strategies to solve the constrained shortest path problem. *Appl. Math. Comput.* 265 (2015), 602 – 618. https://doi.org/10.1016/j.amc.2015. 05.109
- [51] Ning Shi, Shaorui Zhou, Fan Wang, Yi Tao, and Liming Liu. 2017. The multicriteria constrained shortest path problem. *Transportation Research Part E: Logistics and Transportation Review* 101 (2017), 13–29.
- [52] Sabine Storandt. 2012. Route planning for bicycles—exact constrained shortest paths made practical via contraction hierarchy. In ICAPS.
- [53] George Tsaggouris and Christos Zaroliagis. 2009. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. *Theory of Computing Systems* 45, 1 (2009), 162–186.
- [54] Piet Van Mieghem and Fernando A Kuipers. 2003. On the complexity of QoS routing. Computer communications 26, 4 (2003), 376–387.
- [55] Sibo Wang, Wenqing Lin, Yi Yang, Xiaokui Xiao, and Shuigeng Zhou. 2015. Efficient route planning on public transportation networks: A labelling approach. In Proceedings of the 2015 ACM SIGMOD international conference on Management of data. 967–982.
- [56] Sibo Wang, Xiaokui Xiao, Yin Yang, and Wenqing Lin. 2016. Effective indexing for approximate constrained shortest path queries on large road networks. *Proceedings of the VLDB Endowment* 10, 2 (2016), 61–72.

- [57] Yu-Ting Wen, Kae-Jer Cho, Wen-Chih Peng, Jinyoung Yeo, and Seung-won Hwang. 2015. KSTR: Keyword-aware skyline travel route recommendation. In 2015 IEEE international conference on data mining. IEEE, 449–458.
- [58] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. 2014. Path problems in temporal graphs. *Proceedings of the VLDB Endowment* 7, 9 (2014), 721–732.
- [59] Yajun Yang, Hang Zhang, Hong Gao, and Xin Wang. 2021. An efficient index method for the optimal path query over multi-cost networks. World Wide Web (2021), 1–23.
- [60] Pranali Yawalkar and Sayan Ranu. 2019. Route Recommendations on Road Networks for Arbitrary User Preference Functions. In 2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 602–613.
- [61] Jin Y Yen. 1971. Finding the k shortest loopless paths in a network. Management Science 17, 11 (1971), 712–716.
- [62] Mengxuan Zhang, Lei Li, Pingfu Chao, Wen Hua, and Xiaofang Zhou. 2020. Path query processing using typical snapshots in dynamic road networks. In International Conference on Database Systems for Advanced Applications. Springer,

255-271.

- [63] Mengxuan Zhang, Lei Li, Wen Hua, Rui Mao, Pingfu Chao, and Xiaofang Zhou. 2021. Dynamic Hub Labeling for Road Networks. In 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 336–347.
- [64] Mengxuan Zhang, Lei Li, Wen Hua, and Xiaofang Zhou. 2020. Stream Processing of Shortest Path Query in Dynamic Road Networks. *IEEE Transactions on Knowledge and Data Engineering* (2020).
- [65] Mengxuan Zhang, Lei Li, Wen Hua, and Xiaofang Zhou. 2021. Efficient 2hop labeling maintenance in dynamic small-world networks. In 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 133–144.
- [66] Mengxuan Zhang, Lei Li, and Xiaofang Zhou. 2021. An experimental evaluation and guideline for path finding in weighted dynamic network. *Proceedings of the* VLDB Endowment 14, 11 (2021), 2127–2140.
- [67] Xuanyi Zhang, Ziyi Liu, Mengxuan Zhang, and Lei Li. 2021. An Experimental Study on Exact Multi-constraint Shortest Path Finding. In ADC 2021, Dunedin, New Zealand. Springer, 166–179.