Efficient Skyline Keyword-Based Tree Retrieval on Attributed Graphs

Dingming Wu, Zhaofen Zhang, Christian S. Jensen, *Fellow, IEEE*, and Kezhong Lu

***Abstract*—Attributed graphs are graphs, where the vertices have attributes. Such graphs encompass, e.g., social network graph, citation graphs, and knowledge graphs, which have numerous real-world applications. Keyword-based search is a prominent and user-friendly way of querying attributed graphs. One widely used approach to keyword search adopts tree-based query semantics that relies on scoring functions that aggregate distances from a root to keyword-matched vertices. However, it is non-trivial to design scoring functions that capture different users' keyword preferences. This study defines and solves the skyline KTree retrieval problem that combines keyword querying with skyline functionality on attributed graphs. The result of a skyline KTree query is independent of scoring functions. Hence, no matter which keywords are preferred, users can always find their favorite KTrees in a result. To enable efficient skyline KTree retrieval, we propose algorithm FilterRefine that first identifies candidate results and then uses them for search space pruning. Computing distances between keywords and vertices is expensive and dominates the computational cost of FilterRefine. Inspired by subspace skyline query techniques, we convert the skyline KTree retrieval problem into a multi-dimensional subspace skyline problem and propose algorithm MultiDiSkylineOpt. This algorithm is able to reuse skylines in subspaces and uses bounds on all dimensions to accelerate distance computation. Experimental results on real datasets show that a baseline algorithm cannot report results within a 500 second cut-off time, while the proposed algorithms are able to compute results in reasonable time. In particular, MultiDiSkylineOpt is able to efficiently retrieve skyline KTrees on large graphs with millions of nodes and hundreds of millions of edges.**

***Index Terms*—Skyline query, keyword search, attributed graph, query processing.**

I. Introduction

GRAPH-STRUCTURED data occurs in many settings, such as in social networks [1], transportation networks [2], and knowledge graphs [3], and is a rich and valuable source of information. Keyword-based search is a prominent and user-friendly way of retrieving information from schema-free, attributed graphs [4], [5]. Given a set $Q = \{w_i\}$ of query keywords and an attributed graph G, where each vertex has a text attribute, an answer to Q is a subtree [3], [6] or a subgraph [7], [8], [9] extracted from G, such that the vertices in the answer collectively cover part of or all the keywords in Q. In the tree-based query semantics, different scoring functions have been designed to rank answer trees, e.g., the sum of edge weights [6] or the sum of the distances from the root to keyword-matched vertices [3], [10]. However, finding an answer tree that minimizes the sum of the edge weights is NP-complete [11], making this scoring function impractical for large graphs. Hence, this study focuses on retrieving answer trees with designated roots [3], [10] that can be computed in polynomial time.

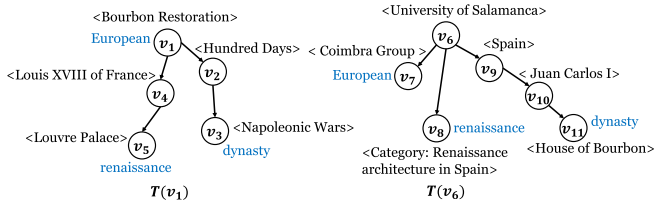


Fig. 1. Two answer trees.

TABLE I
Text Attributes of Vertices

Vertex	Attribute
v_1	**European**, constitutional, bourbon, predecessor, king, country, saint
v_2	king, war, victory, kill, kingdom, event, exile, empire, coalition
v_3	**dynasty**, napoleon, conquer, Portugal, Montenegro, empire
v_4	house, successor, king, gold, France, prince, emperor, leader
v_5	**renaissance**, pyramid, Versailles, classicism, louvre, palais, gothic
v_6	university, institution, Latin, education, charter, operation, Salamanca
v_7	**European**, affiliation, Portugal, executive, group, oldest, association
v_8	**renaissance**, subject, architecture, Spain
v_9	affiliation, allegiance, northwest, venue, region, popularity, country
v_{10}	reign, college, victor, country, parent, Italy, leader, medal, family
v_{11}	**dynasty**, Italian, allegiance, country, member, royal, king, kingdom

Motivation: Given query keywords {European, renaissance, dynasty}, Fig. 1 shows two answer trees $T(v_1)$ and $T(v_6)$ rooted at v_1 and v_6, respectively. Table I shows the text attributes of the vertices in these two trees. In tree $T(v_1)$, vertices v_1, v_3, and v_5 collectively cover the query keywords. In tree $T(v_6)$, vertices v_7, v_8, and v_{11} collectively cover the query keywords. If taking the sum of the distances from the root to all the keyword-matched vertices as the scoring function, tree $T(v_1)$ obtains a score of 4 and is ranked before tree $T(v_6)$, whose score is 5. However, a user who prefers the keyword renaissance may be more interested in $T(v_6)$ than in $T(v_1)$ because renaissance is closer to root v_6 in $T(v_6)$ than to root v_1 in $T(v_1)$. Hence,

Manuscript received 17 April 2023; revised 30 October 2023; accepted 10 April 2024. Date of publication 24 April 2024; date of current version 27 September 2024. This work was supported in part by the Natural Science Foundation of China under Grant 62372308 and in part by the Natural Science Foundation of Guangdong Province of China under Grant 2023A1515011619. Recommended for acceptance by G. Wang. *(Corresponding author: Kezhong Lu.)*

Dingming Wu, Zhaofen Zhang, and Kezhong Lu are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: dingming@szu.edu.cn; zhangzhaofen2021@szu.edu.cn; kzlu@szu.edu.cn).

Christian S. Jensen is with the Department of Computer Science, Aalborg University, 9220 Aalborg, Denmark (e-mail: csj@cs.aau.dk).

Digital Object Identifier 10.1109/TKDE.2024.3388988

when users have diverse preferences, any single scoring function may miss interesting answer trees. Moreover, it is non-trivial to design scoring functions that satisfy all users' preferences.

Proposal of Skyline KTrees: In this study, we adopt the idea of the skyline operator [12] and propose the *skyline keyword-based tree (KTree) retrieval* query on attributed graphs. We call the answer trees mentioned in Fig. 1 KTrees. A KTree is a minimal tree for a set of query keywords. Adopting tree-based query semantics [3], [10], in a KTree, the closer the keywords are to the root, the more relevant the KTree is to the query. For each KTree, we construct a Q-vector, where each dimension refers to the distance of one keyword to the root. A dominance relationship between KTrees is then defined based on the corresponding Q-vectors. A skyline KTree retrieval query returns a set S of KTrees such that the KTrees in S cannot be dominated by other KTrees. For any monotone scoring function that aggregates the distances of the query keywords to the root, the KTree that minimizes that scoring function is included in S. This way, no matter which keyword is preferred, the user can find the best KTree in the result. For instance, in Fig. 1, KTrees $T(v_1)$ and $T(v_6)$ are the skyline KTrees of keywords $\{\mathsf{European}, \mathsf{dynasty}, \mathsf{renaissance}\}$. KTree $T(v_1)$ is better than KTree $T(v_6)$ because the distance of European to root v_1 is shorter than that to root v_6, while KTree $T(v_6)$ is better than KTree $T(v_1)$ because the distance of renaissance to root v_6 is shorter than that to root v_1.

Applications: KTree retrieval queries can be used to access linked data resources represented as RDF (Resource Description Framework) graphs and find entities, relationships, and concepts that satisfy various criteria. In biological graphs, KTree retrieval queries can help researchers identify genes, proteins, and their interactions relevant to different diseases or biological processes. KTree retrieval queries can also help find users, posts, or web pages related to multiple topics in social networks.

Challenges: Skyline KTrees possess two properties that make computations on large attributed graphs challenging. First, skyline KTrees depend on query keywords that become known only when a query is issued. In other words, the dimension values of the Q-vectors are only known when a query is issued. Hence, existing pre-computation based techniques [13], [14], [15], [16] cannot be applied in advance to accelerate subsequent querying. One may consider a two-phase method that first materializes all Q-vectors and then applies an existing skyline computation algorithm. However, the materialization involves extensive explorations of graphs, which is expensive and fails to scale to large graphs. Second, different keyword queries produce different skyline KTrees. If putting all Q-vectors in a $|W|$-dimensional space, where $|W|$ is the number of distinct keywords, a keyword query can be regarded as requesting the skylines in a subspace. However, the number of all possible keyword queries is exponential in $|W|$, which is usually a large number. The resulting very large search space impedes the adoption of existing subspace skyline algorithms [17]. Hence, no practical algorithm exists for skyline KTree retrieval on large attributed graphs.

Contributions: To the best of our knowledge, this is the first study of efficient skyline KTree retrieval. We first propose algorithm FilterRefine. It adopts an α-hop inverted index that stores keyword information in the neighborhoods of all vertices. Easy skyline KTrees and candidate KTrees are first identified from the α-hop inverted index. Then, candidates are eliminated using two pruning strategies. Next, the algorithm computes the distances between query keywords and the remaining candidates. In the end, it finalizes the skyline results by conducting dominance tests on the candidates. Algorithm FilterRefine does not rely on any pre-computation. When queries are issued, the algorithm's pruning strategies effectively reduce the search space and accelerate the query processing.

Next, inspired by subspace skyline algorithms, we convert the problem of skyline KTree retrieval into the problem of multi-dimensional subspace skylines and propose algorithm MultiDiSkylineOpt. The idea is to reuse the skylines in subspaces and derive bounds on all dimensions so that the search space can be pruned effectively. Unlike existing multi-dimensional subspace skyline algorithms that depend on all materialized Q-vectors, algorithm MultiDiSkylineOpt only pre-computes the results in single-dimensional subspaces, which is efficient in terms of both time and space. When queries are issued, the computations of results in multi-dimensional subspaces is accelerated thanks to reused skylines and derived bounds on dimensions. The main contributions of the paper are summarized as follows.

- We defines the problem of skyline KTree retrieval that combines keyword-based retrieval on attributed graphs with skyline functionality.
- We propose algorithm FilterRefine that includes two pruning strategies to reduce the number of candidates.
- Considering skyline KTree retrieval as a multi-dimensional subspace skyline problem, we propose algorithm MultiDiSkylineOpt.
- Extensive experiments are conducted on five real datasets. The findings offer detailed insight into the efficiency of the proposed algorithms.

Outline: The problem of skyline KTree computation on attributed graphs is defined in Section II. Section III presents the filter-and-refinement algorithm. Section IV models skyline KTree retrieval as a multi-dimensional subspace skyline retrieval problem, and proposes algorithm MultiDiSkylineOpt. The experimental evaluation is included in Section V. We review the related work in Section VI and conclude in Section VII.

II. Problem Definition

This section define the concept of skyline keyword-based tree and related concepts. A summary of notations used in the paper is given in Table II.

This study focuses on keyword search on attributed graphs, so only text attributes are considered in the problem definition and proposed solutions. An attributed graph is denoted as $G = (V, E)$, where V is a set of vertices and $E \subseteq V \times V$ is a set of directed edges. Each vertex $v \in V$ is associated with a text attribute $A_v = \{w_i\}$, which is a set of words. Existing tree-based query semantics [4], [5] are described in Definition 1.

Definition 1. Keyword-Based Tree (KTree): Given a set of query keywords $Q = \{w_i\}$ and an attributed graph G, a

TABLE II
NOTATION

Notation	Description
G	An attributed graph
Q	A set of query keywords
w	A keyword
v	A vertex
A_v	The text attribute of vertex v
$T(v)$	A KTree rooted at vertex v
$\vec{v}$	The Q-vector of KTree $T(v)$
$\vec{v}[i]$	The i^{th} dimension of Q-vector $\vec{v}$
$d(v,u)$	The length of the shortest path from v to u
$d_T(v,w)$	The distance from vertex v to keyword w in a KTree
$d_G(v,w)$	The distance from vertex v to keyword w in graph G
$v \sim w$	Vertex v reaches word w in G
$v \nsim w$	Vertex v does not reach word w in G

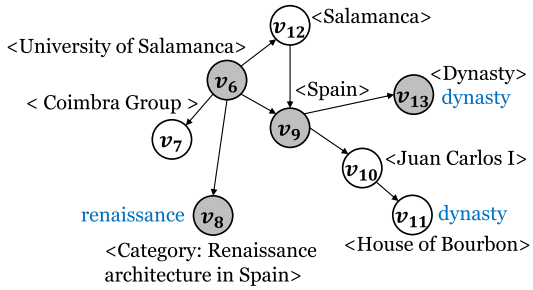


Fig. 2. Attributed graph.

keyword-based tree $T(v) = (V', E')$ is a subtree extracted from G, rooted at vertex v that satisfies two conditions:

1) *Q-cover:* each keyword w_i in Q must be contained in at least one vertex in $T(v)$: $\forall w_i \in Q(\exists v \in V'(w_i \in A_v))$.
2) *Minimal:* any subtree of $T(v)$ is not *Q-cover*.

Given a set Q of l query keywords, based on a fixed order $w_1, w_2, \ldots, w_l$, for each KTree $T(v)$, we construct an l-dimensional Q-vector $\vec{v}$, in which each dimension refers to a keyword in Q, and the value of the i^{th} dimension (corresponding to keyword w_i) in $\vec{v}$ is the distance from root v to w_i in KTree $T(v)$, defined as follows.

$$d_T(v, w_i) = \min_{u \in V' \wedge w_i \in A_u} d(v, u), \tag{1}$$

where $d(v,u)$ is the length of the shortest path from v to u in $T(v)$. When multiple KTrees have the same root v, they are denoted by $T(v^1), T(v^2), \ldots$.

Example 1: Fig. 2 shows an attributed graph for which the text attributes of some of the vertices are included in Table I. Vertices v_{11} and v_{13} match keyword dynasty, and vertex v_8 matches keyword renaissance. The distance from v_6 to dynasty is 2 in the graph. Given keywords: {renaissance, dynasty}, the tree consisting of grey vertices in Fig. 2 is a KTree rooted at v_6. Based on the keyword order (renaissance, dynasty), the Q-vector $\vec{v_6}$ is (1,2).

Definition 2. KTree Dominance: Given a set of keywords Q, let $\mathcal{T}$ be the set of KTrees w.r.t. Q. KTree dominance is a partial order defined on $\mathcal{T}$. KTree $T(v_1)$ *dominates* KTree $T(v_2)$, denoted as $T(v_1) \prec T(v_2)$, if for all i, $\vec{v}_1[i] \leq \vec{v}_2[i]$, and for at least one i, $\vec{v}_1[i] < \vec{v}_2[i]$, where $\vec{v}[i]$ is the ith dimension of Q-vector $\vec{v}$ and $1 \leq i \leq l$.

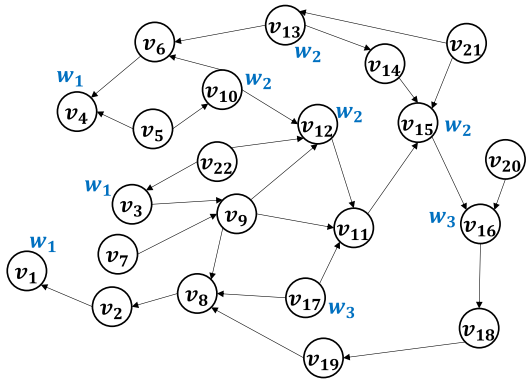


Fig. 3. Running example.

Example 2: Given the ordered keywords (renaissance, dynasty), two KTrees that have the same root v_6 in Fig. 2, i.e., $T(v_6^1)$ consisting of v_6, v_8, v_9, v_{13} and $T(v_6^2)$ consisting of $v_6, v_8, v_9, v_{10}, v_{11}$. The corresponding Q-vectors are $\vec{v}_6^1 = (1,2)$ and $\vec{v}_6^2 = (1,3)$. KTree $T(v_6^1)$ dominates KTree $T(v_6^2)$.

Definition 3. Skyline KTree Retrieval: A skyline KTree retrieval query consists of a set of keywords Q. Given an attributed graph G, let $\mathcal{T}$ contain all the KTrees w.r.t. Q. KTree $T(v)$ is a skyline KTree in $\mathcal{T}$ if and only if it is not dominated by any other KTree in $\mathcal{T}$. The result of query Q on G consists of all the skyline KTrees in $\mathcal{T}$.

III. FILTER-AND-REFINEMENT SOLUTION

This section proposes the filter-and-refinement algorithm for efficiently computing skyline KTrees. We construct an α-hop inverted index that stores the words in the neighborhoods of all vertices in the graph. The idea of the filter-and-refinement algorithm is to first identify easy skyline KTrees from an α-hop inverted index and then eliminate candidates using two pruning strategies.

A. α-Hop Inverted Index

For each vertex v in $G = (V, E)$, the keywords that are α-hop from v are collected via a breadth-first search. These keywords with their corresponding distances are stored in an α-hop inverted index I^α. Specifically, in I^α, each word w is associated with (i) a list of pairs $(v, d_G(v,w))$, where $d_G(v,w) = \min_{u \in V \wedge w \in A_u} d(v,u)$ is the distance from vertex v to word w in G and (ii) the minimum value of $d_G(v,w)$ in the list, denoted by μ. The pairs in the list are ordered non-decreasingly according to $d_G(v,w)$.

Example 3: Fig. 3 shows an attributed graph consisting of 22 vertices. Vertices v_1, v_3, and v_4 match keyword w_1, vertices v_{10}, v_{12}, v_{13}, and v_{15} match keyword w_2, and vertices v_{16} and v_{17} match keyword w_3. Table III shows the α-hop inverted index of the graph in Fig. 3, where $\alpha = 3$.

TABLE III
α-HOP INVERTED INDEX ($\alpha = 3$)

Word	μ	List
w_1	0	$(v_1, 0)$, $(v_3, 0)$, $(v_4, 0)$, $(v_2, 1)$, $(v_5, 1)$, $(v_6, 1)$, $(v_{22}, 1)$, $(v_8, 2)$, $(v_{10}, 2)$, $(v_{13}, 2)$, $(v_9, 3)$, $(v_{17}, 3)$, $(v_{19}, 3)$, $(v_{21}, 3)$
w_2	0	$(v_{10}, 0)$, $(v_{12}, 0)$, $(v_{13}, 0)$, $(v_{15}, 0)$, $(v_5, 1)$, $(v_9, 1)$, $(v_{11}, 1)$, $(v_{14}, 1)$, $(v_{21}, 1)$, $(v_{22}, 1)$, $(v_3, 2)$, $(v_7, 2)$, $(v_{17}, 2)$
w_3	0	$(v_{16}, 0)$, $(v_{17}, 0)$, $(v_{15}, 1)$, $(v_{20}, 1)$, $(v_{11}, 2)$, $(v_{14}, 2)$, $(v_{21}, 2)$, $(v_9, 3)$, $(v_{12}, 3)$, $(v_{13}, 3)$

B. Skyline KTree Computation

Algorithm 1 shows the pseudo code of the proposed algorithm. It consists of four phases. First, skyline KTrees are computed based on the α-hop inverted index. Second, candidates are partitioned based on dimension values, and two pruning strategies are applied to eliminate dominated partitions. Third, in the remaining partitions, each un-materialized vertex is examined. Finally, dominance tests are performed between the found KTrees, and the skyline KTrees of the query are returned. We proceed to elaborate on each part of the algorithm.

Identifying skyline KTrees from I^α (lines 1–16): For each query keyword w, the corresponding $list$ in I^α is retrieved. If any $list$ is empty, the query has no result, and the algorithm terminates. In each $list$, the vertices whose distances are the smallest are added to set $Skyline$, and the remaining vertices in each $list$ are added to set $Candidate$. For any vertex v in $Skyline$, KTree $T(v)$ cannot be dominated by any other KTree $T(v')$, $v' \in V \setminus Skyline$, since at least one dimension of $\vec{v}$ has the minimum value. In set $Skyline$, the Q-vectors of some vertices may not be materialized because the distances from these vertices to some query keywords exceed α and are not indexed in I^α. To materialize these Q-vectors, reachability tests based on TF-Labels [18] are first conducted, and their distances to query keywords are then computed by issuing shortest-path queries [19]. In graph G, not all vertices may connect to a query keyword. Thus, conducting reachability tests before shortest-path queries saves computational costs for unreachable query keywords. Next, function Evaluate[1] in the BUS algorithm [17] is applied on set $Skyline$ to eliminate dominated KTrees, so that the rest are in the skyline.

Value-Based Partitioning (line 17): Given a query, the Q-vectors of vertices are integer vectors, and the ranges of the vector dimensions are small. Based on this observation, we partition the vertices in set $Candidate$ based on the dimension values of their Q-vectors. Specifically, in each partition P, the distances from all vertices to the same keyword are the same, i.e., $\forall w \in Q\ (\forall v, v' \in P\ (d_G(v, w) = d_G(v', w)))$. We create a dummy Q-vector p^* to represent all Q-vectors in partition P such that for any $v \in P$ and for each dimension i, $\vec{p^*}[i] = \vec{v}[i]$. This way, partition-wise dominance testing is much more efficient than vertex-wise dominance testing.

Pruning Strategies (lines 18–25): We perform two-phase dominance testing on the partitions. The first phase compares each partition with the skyline KTrees in set $Skyline$. If the dummy vector p^* of partition P is dominated by one skyline KTree, all vertices in partition P are pruned. In the second phase, dominance tests are performed between partitions. If the dummy vector p_i^* of partition P_i is dominated by the dummy vector p_j^* of partition P_j, all vertices in partition P_i are pruned.

Distance Materialization (lines 26–31): For each surviving partition P (neither dominated by any skyline KTree in $Skyline$ nor dominated by any other partition), if its dummy vector is materialized, all the vertices in P are added to $Skyline$. Otherwise, for each vertex in P, we compute its distances to the query keywords by issuing shortest-path queries.

Example 4: To illustrate the filter-and-refinement algorithm, consider the attributed graph in Fig. 3 and the α-hop inverted index in Table III. Given the ordered query keywords (w_1, w_2, w_3), we access the α-hop inverted index and add vertices v_1, v_3, v_4, v_{10}, v_{12}, v_{13}, v_{15}, v_{16}, and v_{17} to set $Skyline$ and add vertices v_2, v_5, v_6, v_7, v_8, v_9, v_{11}, v_{14}, v_{19}, v_{20}, v_{21}, and v_{22} to set $Candidate$. In $Skyline$, vertices v_1, v_4, and v_{16} cannot reach all query keywords and thus are removed, leaving six vertices

Algorithm 1: FilterRefine(Query Q, Index I_α, Graph G).

```
1:  Skyline ← ∅; Candidate ← ∅;
2:  for each w in Q do
3:    list ← I_α.Get(w);
4:    if list = ∅ then
5:      Return ∅;
6:    for each entry (v, d) in list do
7:      if d is the smallest then
8:        Remove (v, d) from list;
9:        Add/Update v to/in Skyline;
10:   Candidate ← the vertices left in list;
11: for each v ∈ Skyline do
12:   if v is not materialized then
13:     if ReachabilityTest(v, Q, G) is FALSE then
14:       Remove v;
15:     Materialize(v, Q, G);
16: Evaluate(Skyline);
17: P ← Partition(Candidate);
18: for each partition P ∈ P do ▷Skyline-based pruning.
19:   if ∃v ∈ Skyline (T(v) ≺ T(p)) then
20:     P is pruned and removed;
21:     Continue;
22: for each partition P_i ∈ P do ▷Inter-partition pruning.
23:   if ∃P_j ∈ P (p_j^* is materialized ∧ T(p_j^*) ≺ T(p_i^*))
    then
24:     P_i is pruned and removed;
25:     Continue;
26:   if p_i^* is not materialized then
27:     for each v ∈ P_i do
28:       if ReachabilityTest(v, Q, G) then
29:         Materialize v based on Q;
30:   else
31:     Add all the vertices in P_i to Skyline;
32: Add all the vertices in P to Skyline;
33: Evaluate(Skyline);
34: Return Skyline;
```

[1] Evaluate optimizes dominance testing by adopting a filtering function.

TABLE IV
VERTEX PARTITIONS

Partition	Vertices	Dummy Q-vector
P_1	v_2, v_6	$\vec{p}_1^* = (1, 4, 4)$
P_2	v_8	$\vec{p}_2^* = (2, 4, 4)$
P_3	v_{19}	$\vec{p}_3^* = (3, 4, 4)$
P_4	v_5, v_{22}	$\vec{p}_4^* = (1, 1, 4)$
P_5	v_9	$\vec{p}_5^* = (3, 1, 3)$
P_6	v_{21}	$\vec{p}_6^* = (3, 1, 2)$
P_7	v_{11}, v_{14}	$\vec{p}_7^* = (4, 1, 2)$
P_8	v_{20}	$\vec{p}_8^* = (4, 0, 4)$
P_9	v_7	$\vec{p}_9^* = (4, 2, 4)$

in $Skyline$, whose Q-vectors are $\vec{v}_3 = (0, 2, 4)$, $\vec{v}_{10} = (2, 0, 4)$, $\vec{v}_{12} = (8, 0, 3)$, $\vec{v}_{13} = (2, 0, 3)$, $\vec{v}_{15} = (6, 0, 1)$, $\vec{v}_{17} = (3, 2, 0)$. Next, function Evaluate eliminates $\vec{v}_{10}$ and $\vec{v}_{12}$ because they are dominated by $\vec{v}_{13}$ and $\vec{v}_{15}$, respectively. At this point, we have four skyline KTrees $T(v_3)$, $T(v_{13})$, $T(v_{15})$, and $T(v_{17})$. Then, the vertices in $Candidate$ are grouped into 9 partitions based on the dimension values, shown in Table IV. Since $\alpha = 3$, for a vertex whose distance to a keyword is not in the index, we temporally set the corresponding dimension value to 4. Next, skyline-based pruning is applied and partitions P_1, P_2, P_3, P_5, P_8, and P_9 are dominated and are removed. Then, inter-partition pruning is applied, where partition P_7 is dominated by P_6. After that, the vertices in partitions P_4 and P_6 are examined, finding that v_5 is dominated by v_{22}. Finally, skyline KTrees $T(v_3)$, $T(v_{13})$, $T(v_{15})$, $T(v_{17})$, $T(v_{21})$, and $T(v_{22})$ are returned.

Time and Space Complexity: Given a query Q with $|Q|$ keywords, there are at most $|V|$ KTrees in the graph. Algorithm 1 first selects s ($< |V|$) vertices and adds them to set $Skyline$, taking time $O(|V|)$. Next, for each of these s vertices, the TF-Labels algorithm tests reachability to the query keywords, taking time $O(\log L)$, where L is the label size, and then at most $|Q|$ traditional shortest-path queries are issued to materialize the corresponding KTree, taking time $O(|V| + |E|)$. Next, the BUS algorithm is applied to eliminate dominated KTrees from $Skyline$, taking time $O(s^2 \cdot |Q|)$. Then, let c ($< |V|$) be the number of candidate vertices that are divided into $|P|$ partitions, which is $O(c \cdot |Q|)$. Skyline-based pruning takes $O(s \cdot |P| \cdot |Q|)$ time, and inter-partition pruning takes $O(|P|^2 \cdot |Q|)$ time. Next, for all vertices in the surviving partitions, it takes $O(c \cdot \log L \cdot |Q|)$ time in the worst case to perform reachability testing, and it takes $O(c \cdot s \cdot |V| \cdot |Q|)$ time to materialize the distances to the keywords. Finally, the dominance tests between the found KTrees take $O(c^2 \cdot |Q|)$ time. Overall, the time complexity of Algorithm 1 is $O(|V| + s \cdot (\log L + |V| + |E|) \cdot |Q| + (s^2 + c^2 + |P|^2 + c + s \cdot |P| + |V|^2 + c \cdot s \cdot |V|) \cdot |Q| + c \cdot \log L \cdot |Q|) \approx O(s \cdot (|V| + |E|) + |V|^2)$. The space complexity of Algorithm 1 is $O(|V|)$.

IV. SUBSPACE BASED SOLUTION

Given an attributed graph $G = (V, E)$, the vocabulary W of G is the union of the text attributes of all the vertex in V: $W = \cup_{v \in V} A_v$. This section tackles the skyline KTree retrieval problem in a $|W|$-dimensional space $\mathcal{S}_W$, where each dimension represents a word in W. Each vertex $v \in V$ has a vector $\vec{v}$ in space $\mathcal{S}_W$. The value of a dimension (corresponding to word w_i) in $\vec{v}$ is the distance from v to w_i in G, i.e., $d_G(v, w_i)$. Given a subset $X \subseteq W$, we use $\vec{v}_X$ to denote the projection of v into subspace $\mathcal{S}_X$.

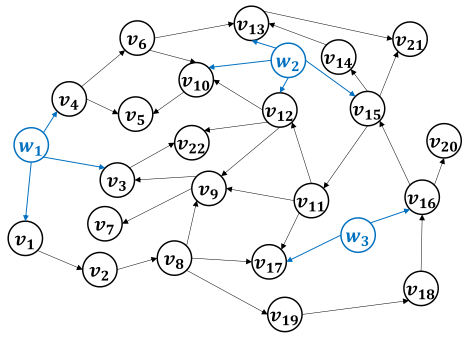


Fig. 4. Reverse graph.

Definition 4: Given a vertex v and a word w, if a path exists in G from v to u ($\in V$) such that $w \in A_u$ then vertex v reaches word w, denoted by $v \sim w$. Otherwise, vertex v does not reach word w, denoted by $v \nsim w$.

Definition 5: Given a subspace $\mathcal{S}_Y \subseteq \mathcal{S}_W$, for a vertex $v \in V$, if $\forall w \in Y (v \sim w)$, then its projected vector $\vec{v}_Y$ exists in subspace $\mathcal{S}_Y$, denoted by $\vec{v}_Y \in \mathcal{S}_Y$; if $\exists w \in Y (v \nsim w)$, then its projection vector $\vec{v}_Y$ does not exist in subspace $\mathcal{S}_Y$, denoted by $\vec{v}_Y \notin \mathcal{S}_Y$.

Given a set Q ($\subseteq W$) of l keywords, each KTree $T(v)$ has a corresponding l-dimensional projection vector in subspace $\mathcal{S}_Q$ ($\subseteq \mathcal{S}_W$), denoted by $\vec{v}_Q$. Thus, based on Definition 3, the result of a skyline KTree retrieval query Q can be regarded as the skyline vectors in subspace $\mathcal{S}_Q$. When the context is clear, we use space instead of subspace for brevity.

Example 5: Consider the attributed graph in Fig. 3. Given query $Q = (w_1, w_2, w_3)$, the projection vector of vertex v_3 into space $\mathcal{S}_Q$ is $\vec{v}_3 = (0, 2, 4)$. The projection vector of vertex v_{16} does not exist in space $\mathcal{S}_Q$ because $v_{16} \nsim w_1$ and $v_{16} \nsim w_2$.

We proceed to show how to materialize skyline vectors in all single-dimensional subspaces. Space $\mathcal{S}_Q$ for a given query is decomposed into several subspaces. The skyline result in $\mathcal{S}_Q$ is then computed by utilizing the skyline vectors in the subspaces of $\mathcal{S}_Q$.

A. Skyline Computation in Single-Dimensional Subspaces

Given an attributed graph G, we construct its reverse graph G^R with word vertices injected as follows. A word vertex v_w for each word $w \in W$ is created and inserted into G. For each vertex $v \in V$, directed edge (v, v_w) is added into G, if $w \in A_v$. After injecting all word vertices into G, we create a reverse graph G^R that is on the same set of vertices of G with all of the edges reversed. Fig. 4 shows the reverse graph corresponding to the attributed graph in Fig. 3.

Algorithm 2 shows the pseudo code of the skyline computation in a single-dimensional subspace. In graph G^R, for a word $w \in W$, the neighbor vertices of vertex v_w are the

Algorithm 2: SingleDiSkyline(Set W, Graph G^R).

```
1: Initialize array Sky[ ];
2: for each word w in W do
3:    Sky[w] ← getNeighbors(G^R, v_w);
4: return Sky[ ];
```

vertices whose distances to w are equal to 0. Thus, the projection vectors $\vec{v}_w$ of these neighbor vertices are the skyline vectors in single-dimensional subspace $\mathcal{S}_w$. The skyline vectors of all the single-dimensional subspaces are stored for the skyline computation in multi-dimensional subspaces.

B. Skyline Computation in Multi-Dimensional Subspaces

Given a set Q of l keywords, we recursively decompose space $\mathcal{S}_Q$ into an $(l-1)$-dimensional subspace $\mathcal{S}_X$ and a single-dimensional subspace $\mathcal{S}_Y$. The skyline vectors in these subspaces facilitate skyline computation in space $\mathcal{S}_Q$. The skyline vectors in $\mathcal{S}_Q$ can be divided into two groups. The first group contains some of the skyline vectors in subspaces $\mathcal{S}_X$ and $\mathcal{S}_Y$. The skyline vectors in the second group are vectors from $\mathcal{S}_Q$. We proceed to introduce three lemmas that help finding these two groups of skyline vectors efficiently and then present algorithm MultiDiSkyline.

Given a skyline vector $\vec{v}_X$ in a space $\mathcal{S}_X$, Lemma 1 gives the condition that the corresponding projection vector of v must satisfy to be a skyline vector in a superspace of $\mathcal{S}_X$.

Lemma 1: Let $Sky(X)$ include the skyline vectors in subspace $\mathcal{S}_X$ and $Sky(X \cup Y)$ include the skyline vectors in space $\mathcal{S}_{X\cup Y}$. Given a skyline vector $\vec{v}_X \in Sky(X)$, if $\vec{v}_Y \in \mathcal{S}_Y$ and $\nexists \vec{u}_X \in Sky(X)(\vec{u}_X = \vec{v}_X)$ then projection vector $\vec{v}_{X\cup Y} \in Sky(X \cup Y)$.

Proof: Suppose vector $\vec{v}_{X\cup Y} \notin Sky(X \cup Y)$, which implies that there exists a vector $\vec{v}^*_{X\cup Y}$ in space $Sky(X \cup Y)$, such that $\vec{v}^*_{X\cup Y} \prec \vec{v}_{X\cup Y}$. Then, we have $\vec{v}^*_X[i] \leq \vec{v}_X[i], 1 \leq i \leq |X|$. According to the condition $\nexists \vec{u}_X \in Sky(X)(\vec{u}_X = \vec{v}_X)$, we can derive that for at least one i, $\vec{v}^*_X[i] < \vec{v}_X[i]$. Then, we have $\vec{v}^*_X \prec \vec{v}_X$, which contradicts the condition that $\vec{v}_X \in Sky(X)$. Hence, the lemma holds. □

Example 6: Let $X = (w_1, w_2)$ and $Y = (w_3)$. In Fig. 3, $\vec{v}_3 = (0, 2)$ is a skyline vector in space $\mathcal{S}_X$. In space $\mathcal{S}_Y$, $\vec{v}_3 = (4)$. No other vector in space $\mathcal{S}_X$ equals $\vec{v}_3$. Hence, $\vec{v}_3 = (0, 2, 4)$ is a skyline vector in space $\mathcal{S}_{X\cup Y}$.

Let $\mathcal{S}_X$ be a k-dimensional space. Based on the skyline vectors in $\mathcal{S}_X$, Lemma 2 derives the condition that a vector must satisfy to not be a skyline vector in a $(k+1)$-dimensional superspace of $\mathcal{S}_X$.

Lemma 2: Given space $\mathcal{S}_X$ and its skyline vector set $Sky(X)$. Let $\mathcal{S}_Y$ be a single-dimensional space. Define $max_y = \max_{\vec{v}\in Sky(X)}\{\vec{v}[Y]\}$, where $\vec{v}[Y]$ refers to the value in dimension Y. A vector $\vec{v} \in \mathcal{S}_{X\cup Y}$ and $\vec{v} \notin Sky(X)$ is not a skyline vector in space $\mathcal{S}_{X\cup Y}$, if $\vec{v}[Y] \geq max_y$.

Proof: Given a vector $\vec{v} \in \mathcal{S}_{X\cup Y}$ and $\vec{v} \notin Sky(X)$, let $\vec{u}$ be a skyline vector in $Sky(X)$ such that $\vec{u}_X \prec \vec{v}_X$. Based on the condition that $\vec{u}[Y] \leq max_y \leq \vec{v}_Y$, we can derive $\vec{u} \prec \vec{v}$ in space $\mathcal{S}_{X\cup Y}$. Hence, $\vec{v}$ is not a skyline vector in space $\mathcal{S}_{X\cup Y}$. □

Example 7: Consider the graph in Fig. 3. Let $X = (w_1, w_2)$ and $Y = (w_3)$. The skyline vectors in space $\mathcal{S}_X$ are v_3, v_5, v_{10}, v_{13}, and v_{22}. Since $\vec{v}_5 = \vec{v}_{22}$ in space $\mathcal{S}_X$, according to Lemma 1, in space $\mathcal{S}_{X\cup Y}$, $\vec{v}_{22} \prec \vec{v}_5$. Then, we have $max_y = v_{22}[Y] = 4$. Vector $\vec{v}_7$ is not a skyline vector in space $\mathcal{S}_{X\cup Y}$ because $\vec{v}_7[Y] = 4 \geq max_y$.

For the vertices that cannot be ruled out by Lemma 2, we need to compute their distances to each keyword by exploring the graph which is expensive. To reduce the cost, Lemma 3 offers a *bound* that helps eliminating vertices during graph traversals.

Lemma 3: Given vector $\vec{u} \in \mathcal{S}_{X\cup Y}$, let S be a set of vectors in space $\mathcal{S}_{X\cup Y}$ such that $\forall \vec{v} \in S(\vec{u}[Y] > \vec{v}[Y])$. Define $bound = \min_{\vec{v}\in S} \max_{1\leq i\leq |X|} \vec{v}[i]$. Vector $\vec{u}$ is not a skyline vector in space $\mathcal{S}_{X\cup Y}$ if $\vec{u}[i] \geq bound, 1 \leq i \leq |X|$.

Proof: Based on the condition that $\exists \vec{v} \in S$ such that $\vec{u}[Y] > \vec{v}[Y]$ and $\vec{u}[i] \geq bound \geq \vec{v}[i], 1 \leq i \leq |X|$, we can derive $\vec{v} \prec \vec{u}$. Hence, $\vec{u}$ is not a skyline vector in space $\mathcal{S}_{X\cup Y}$. □

Example 8: Consider the graph in Fig. 3. Let $X = (w_1, w_2)$, $Y = (w_3)$, and $S = \{\vec{v}_{13} = (2, 0, 3), \vec{v}_{17} = (3, 2, 0)\}$. Then, we have $bound = 2$. Hence, $\vec{v}_7 = (4, 2, 4)$ is not a skyline vector in space $\mathcal{S}_{X\cup Y}$.

Algorithm MultiDiSkyline*:* Algorithm 3 shows the pseudo code of skyline computation in space $\mathcal{S}_{X\cup Y}$ by utilizing the skyline vectors in subspaces $\mathcal{S}_X$ and $\mathcal{S}_Y$, where $\mathcal{S}_Y$ is a single-dimensional space. First (lines 3–6), it reuses some of the skyline vectors in $Sky(X)$ and $Sky(Y)$. Based on Lemma 1, it eliminates skyline vectors in $Sky(X)$ and $Sky(Y)$ that are not skyline vectors in space $\mathcal{S}_{X\cup Y}$ by calling function Refine (algorithm 4). Then, the remaining skyline vectors in subspaces $\mathcal{S}_X$ and $\mathcal{S}_Y$ are added to the result skyline set Sky. Next (lines 9–14), the vectors in space $\mathcal{S}_{X\cup Y}$ are considered in ascending order in dimension Y. According to Lemma 2, it computes max_y (line 7) which is used to prune the search space (lines 13 and 14), i.e., vectors are not skyline vectors in space $\mathcal{S}_{X\cup Y}$ if the values in dimension Y are at least max_y. For the vertices that are not pruned (lines 16–18), the algorithm calls function BFS (Algorithm 5), which conducts a breadth-first search on the graph to compute the distances from these vertices to the query keywords. According to Lemma 3, it uses *bound* (computed in line 15) to stop the exploration on the graph when a vertex is too far from a keyword to become a skyline vector. Finally (lines 19–22), dominance tests are performed on the vectors in set $Cand$, and the result skylines are returned.

Time and Space Complexity: Given a query Q with $|Q|$ keywords, let X and Y be the number of skylines in subspaces $\mathcal{S}_X$ and $\mathcal{S}_Y$, respectively. Algorithm 3 first tests whether the skylines in $\mathcal{S}_X$ ($\mathcal{S}_Y$) exist in subspace $\mathcal{S}_Y$ ($\mathcal{S}_X$), having complexity $O((X+Y)\cdot \log L)$, and then using shortest-path queries to materialize the corresponding KTrees, having complexity $O((X+Y)\cdot(|V|+|E|))$. Next, it takes $O(X^2)$ and $O(Y^2)$ time to remove dominated vectors in subspaces $\mathcal{S}_X$ and $\mathcal{S}_Y$, respectively. Let C_Y be the vectors whose Y dimension values are less than max_y. In the worst case, it takes $O(C_Y \cdot (|V|+|E|))$ time to determine whether these candidates are skylines in line 16. Overall, the time complexity of

Algorithm 3: MultiDiSkyline(Graph G, Keywords X, Keyword Y, Set $Sky(X)$, Set $Sky(Y)$, Set Sky).

$Sky \leftarrow \emptyset$;
$Cand \leftarrow \emptyset$;
Refine$(Sky(X), \mathcal{S}_Y, G)$; ▷Lemma 1;
Refine$(Sky(Y), \mathcal{S}_X, G)$; ▷Lemma 1;
Add $Sky(X)$ to Sky;
Add $Sky(Y)$ to Sky;
$max_y \leftarrow \max_{\vec{v} \in Sky(X)}\{\vec{v}[Y]\}$;
$S \leftarrow Sky(Y)$;
while TRUE **do**
  Access vector $\vec{v}$ in ascending order at dimension Y;
  if $\vec{v} \in Sky$ **then**
    continue;
  if $\vec{v}[Y] = max_y$ **then**
    break; ▷Lemma 2;
  $bound \leftarrow \min_{\vec{v} \in S} \max_{1 \le i \le |X|} \vec{v}[i]$;
  if $\mathbf{BFS}(G, v, bound)$ **then** ▷Lemma 3;
    Add $\vec{v}$ to $Cand$;
    Add $\vec{v}$ to S;
for each vector $\vec{v}$ in $Cand$ **do**
  if $\nexists \vec{u} \in Sky \nexists \vec{v'} \in Cand(\vec{u} \prec \vec{v} \wedge \vec{v'} \prec \vec{v})$ **then**
    Add $\vec{v}$ to Sky;
return Sky;

Algorithm 4: Refine(Skyline Vectors $Sky(X)$, Subspace $\mathcal{S}_Y$, Graph G).

for each vector $\vec{v}_X$ in $Sky(X)$ **do**
  if $\vec{v}_Y \notin \mathcal{S}_Y$ **then**
    Remove $\vec{v}_X$ from $Sky(X)$;
  else if $\exists \vec{u}_X \in Sky(X)(\vec{u}_X = \vec{v}_X \wedge \vec{u}_Y \prec \vec{v}_Y)$ **then**
    Remove $\vec{v}_X$ from $Sky(X)$;
return $Sky(X)$;

Algorithm 5: BFS(Graph G, Vertex v, Int $bound$, Words X).

$Queue \leftarrow \emptyset$;
$Queue$.**enqueue**(v);
while $Queue \neq \emptyset$ **do**
  Vertex $u \leftarrow Queue$.**dequeue**$()$;
  if $\exists w_i \in X(w_i \in A_u)$ **then**
    $\vec{v}[i] \leftarrow d_G(v, w_i)$;
    if all words in X have been encountered **then**
      return TRUE;
  if $d(v, u) < bound$ **then**
    for each adjacent vertex u' of u **do**
      $Queue$.**enqueue**(u');
if no word in X has been encountered **then**
  return FALSE;
else
  return TRUE;

Algorithm 6: MultiDiSkylineOpt(Graph G, Keywords X, Keyword Y, Set $Sky(X)$, Set $Sky(Y)$, Set Sky).

$Sky \leftarrow \emptyset$;
$Cand \leftarrow \emptyset$;
Refine$(Sky(X), \mathcal{S}_Y, G)$; ▷Lemma 1;
Refine$(Sky(Y), \mathcal{S}_X, G)$; ▷Lemma 1;
Add $Sky(X)$ to Sky;
Add $Sky(Y)$ to Sky;
$max_y \leftarrow \max_{\vec{v} \in Sky(X)}\{\vec{v}[Y]\}$;
$S \leftarrow Sky(Y)$;
$bound \leftarrow \min_{\vec{v} \in S} \max_{1 \le i \le |X|} \vec{v}[i]$;
reverseBFS$(G, max_y, bound, X, Y, Sky, Cand)$;
for each vector $\vec{v}$ in $Cand$ **do**
  if $\nexists \vec{u} \in Sky \nexists \vec{v'} \in Cand(\vec{u} \prec \vec{v} \wedge \vec{v'} \prec \vec{v})$ **then**
    Add $\vec{v}$ to Sky;
return Sky;

Algorithm 3 is $O((X+Y) \cdot (|V|+|E|) + C_Y \cdot (|V|+|E|))$. The space complexity is $O(X+Y+C_Y)$.

Algorithm MultiDiSkylineOpt*:* In algorithm MultiDiSkyline, function BFS is conducted on each candidate vertex, which has high cost when the candidate set is large. To reduce this cost, we propose algorithm MultiDiSkylineOpt that performs reverse BFS exploration on the graph starting from keyword vertices instead of candidate vertices. Algorithm 6 shows the pseudo code. First, it reuses some of the skyline vectors in subspaces $\mathcal{S}_X$ and $\mathcal{S}_Y$ based on Lemma 1. Next, it computes max_y and $bound$. After that, function reverseBFS (Algorithm 7) is called to determine the distances from candidate vertices to the keywords in X. It utilizes Lemmas 2 and 3 to eliminate some of the vertices in the neighborhoods of the keyword vertices. Specifically, it first collects the set of vertices whose distances to keyword Y are less than max_y. According to Lemma 2, the vertices encountered during the exploration on the graph that do not belong to this set are non-skyline vertices. According to Lemma 3, only the vertices whose distances are smaller than $bound$ are explored further.

Time and Space Complexity: Algorithm 6 differs from Algorithm 3 in the traversal direction when computing the distances between vertices and keywords. In the worst case, the time and space complexities of Algorithm 6 are the same as for Algorithm 3.

Example 9: Consider the graph in Fig. 3. Let $X = (w_1, w_2)$ and $Y = (w_3)$. We have $Sky(X) = \{\vec{v}_3, \vec{v}_5, \vec{v}_{10}, \vec{v}_{13}, \vec{v}_{22}\}$ and $Sky(Y) = \{\vec{v}_{16}, \vec{v}_{17}\}$. According to Lemma 1, $\vec{v}_{16}$ is removed from $Sky(Y)$ because v_{16} cannot reach w_1 and w_2. And in space $\mathcal{S}_{X \cup Y}$, we have $\vec{v}_{22} \prec \vec{v}_5$ and $\vec{v}_{13} \prec \vec{v}_{10}$, so $\vec{v}_5$ and $\vec{v}_{10}$ are removed from $Sky(X)$. Then, we derive $Sky = \{\vec{v}_3, \vec{v}_{13}, \vec{v}_{17}, \vec{v}_{22}\}$, $max_y = 4$ and $bound = 3$. Next, in function reverseBFS, ht contains candidate vertices v_9, v_{11}, v_{12}, v_{14}, v_{15}, v_{20}, and v_{21}, whose distances to keyword w_3 are less than 4. A priority queue is used to organize the vertices to be visited during the explorations starting from keyword vertices w_1 and w_2. Pairs (v_1, w_1), (v_3, w_1), (v_4, w_1), (v_{10}, w_2), (v_{12}, w_2), (v_{13}, w_2), and (v_{15}, w_2) are first added to the queue. After these pairs are all dequeued, vertices v_{12} and v_{15} are found in ht and are added to set $Cand$. The adjacent vertices of dequeued vertices

Algorithm 7: ReverseBFS(Graph G, Int max_y, Int $bound$, Keywords X, Keyword Y, Set Sky, Set $Cand$).

```
1:  Hashtable ht ← collect(G, Y, max_y, Sky);
2:  PriorityQueue Queue ← ∅;
3:  for each word w_i in X do
4:    for each adjacent vertex v of w_i do
5:      Queue.enqueue(v, w_i);        ▷w_i is the label of v.
6:  while Queue ≠ ∅ do
7:    Vertex v ← Queue.dequeue();
8:    if v⃗[Y] ← ht.get(v) then                ▷Lemma 2;
9:      if v has label w_i then
10:       v⃗[i] ← d_G(v, w_i);
11:       Add/Update v to/in Cand;
12:   if v has label w_i and d_G(v, w_i) < bound then
13:     for each adjacent vertex u of v do
14:       Queue.enqueue(u, w_i);          ▷Lemma 3;
```

whose distances to either w_1 or w_2 is less than 3 are added to the priority queue and are considered in the next iteration. When the queue is exhausted, set $Cand$ contains vertices v_{12}, v_{15}, v_9, v_{11}, v_{14}, and v_{21}. In the end, dominance testing is performed on these candidate vertices, and the result skyline is $Sky = \{\vec{v}_3, \vec{v}_{13}, \vec{v}_{15}, \vec{v}_{17}, \vec{v}_{21}, \vec{v}_{22}\}$.

V. Experimental Evaluation

This section conducts evaluations on five real datasets and provides insight into the efficiency of the proposed algorithms.

A. Setup

Datasets: The proposed algorithms are evaluated on five real datasets: DBpedia,[2] Yago,[3] DBLP [20], LastFm [21], and CiteSeer.[4] We conducted preprocessing and kept the largest single weakly connected component without self-loops in each dataset. Table V shows statistics on the five datasets, including the number of vertices and edges, the number of distinct words, the average number of single-dimensional skylines, and the space needed for storing them.

Workloads: Generating query keywords totally at random may yield empty skyline results. Therefore, we generate meaningful queries by following the keyword distributions of the datasets. To generate a query, we randomly select a vertex v in the graph and then explore the graph from v in BFS manner and randomly select at least $|Q|/2$ and at most $|Q| \cdot factor$ vertices that are reachable from v ($factor \geq 1$). If fewer than $|Q|/2$ vertices are reachable from v, v is discarded to avoid the case that the subgraph around v is too limited. In this case, we randomly select another vertex and repeat the process. Among the selected $[|Q|/2, |Q| \cdot factor]$ vertices, we randomly choose at most $|Q|$ vertices, and $|Q|$ keywords are randomly extracted from the distinct words in the attributes of these vertices as the query keywords. Intuitively, a skyline either (i) has small values in all dimensions or (ii) has small values in some dimensions and large values in other dimensions. Hence, by tuning $factor$, we generate two types of workloads: Workload I (WL I) and Workload II (WL II) to capture the characteristics mentioned above. For each type of workload, we obtain 50 2-keyword queries, 50 3-keyword queries, and 20 4-keyword queries, where each query has a non-empty result. Fewer 4-keyword queries are considered because many such queries return empty results on all datasets. Table VI shows the average largest dimension value in the skylines of the two types of workloads on each dataset. We do not have 4-keyword queries in WL I because it is difficult to obtain skylines whose four dimension values are all small in these datasets. Table VII shows the average number of skyline KTrees of the queries in each workload.

Algorithms: To be covered in Section VI, existing algorithms are designed for different types of skyline queries and thus existing algorithms cannot be adopted directly to compute skyline KTrees. Thus, we construct algorithm Baseline by extending a state-of-the-art algorithm: it first computes the distances from all the vertices to each query keyword and then finds skylines using algorithm SaLSa [22]. To assess the efficiency of our proposals, we report the performance of Baseline, FilterRefine (Algorithm 1), and MultiDiSkylineOpt (Algorithm 6).

Settings: All algorithms are implemented in C++ and are compiled using GNU GCC 6.5.0. Experiments are conducted on a server with an Intel Xeon 2.00 GHz CPU, 2.0 TB main memory, and running Linux (ubuntu Linux 16.04, 64 bits). We set the cut-off time to 500 seconds. If an algorithm runs out of memory or does not complete within the cut-off time, its execution time is marked as "INF". On each dataset and for each algorithm, we report the average running time per query. By default, parameter α in algorithm FilterRefine is set to 3, and we use the workloads of 3-keyword queries.

B. Main Results

Table VIII reports the average running time and the average number of candidates per query for algorithms Baseline, FilterRefine, and MultiDiSkylineOpt on the five datasets. Baseline incurs substantial time on graph explorations to compute the distances of all vertices to each query keyword and cannot report results within the cut-off time on all datasets. Baseline takes all vertices in the graph as candidates, so it considers many more candidates than do the other algorithms. Hence, Baseline is an impractical algorithm and is excluded from subsequent experiments. FilterRefine first gets skylines from the α-hop inverted index and then prunes candidates using derived bounds. For each unpruned candidate vertex, it computes its distances to the query keywords. We observe that the average number of candidates per query in FilterRefine is smaller than that in MultiDiSkylineOpt, but in most cases, the average running time of FilterRefine is higher than that of MultiDiSkylineOpt. This is because the keyword distance computation in FilterRefine is very expensive, which dominates the computational cost. This result also demonstrates that the derived bounds used in function reverseBFS in algorithm MultiDiSkylineOpt are effective at pruning non-skyline KTrees when traversing the graph. On

[2]https://www.dbpedia.org
[3]https://yago-knowledge.org
[4]http://citeseerx.ist.psu.edu

TABLE V
STATISTICS OF DATASETS

Dataset	$\|V\|$	$\|E\|$	#distinct words	Average number of single-dimensional skylines	Storage of single-dimensional skylines
DBpedia	8,099,624	165,262,890	2,926,994	56.46	1.24 GB
Yago	8,091,094	26,443,802	3,373,803	7.84	247 MB
DBLP	4,348,701	71,757,557	6,872,219	9.77	597 MB
LastFm	250,329	101,384,797	3,861,852	26.04	689 MB
CiteSeer	371,240	13,299,356	197,944	67.19	87 MB

TABLE VI
AVERAGE LARGEST DIMENSION VALUE IN SKYLINES

Dataset	2-keyword queries		3-keyword queries		4-keyword queries
	WL I	WL II	WL I	WL II	WL II
DBpedia	2.55	7.60	2.75	9.90	9.85
Yago	2.35	8.30	2.85	10.90	8.30
DBLP	2.2 0	8.20	2.75	11.80	18.50
LastFm	2.50	8.25	2.25	10.65	12.70
CiteSeer	2.25	6.05	2.70	5.15	5.25

TABLE VII
AVERAGE NUMBER OF SKYLINE KTREES

Dataset	2-keyword queries		3-keyword queries		4-keyword queries
	WL I	WL II	WL I	WL II	WL II
DBpedia	15.60	22.15	5.75	4.65	5.60
Yago	12.50	17.20	10.50	8.75	8.20
DBLP	8.60	6.55	7.70	6.30	7.15
LastFm	4.90	12.75	5.65	10.15	9.95
CiteSeer	9.40	22.55	12.50	12.20	7.90

DBpedia (WL II), FilterRefine is faster than MultiDiSkylineOpt. This is because of the large size of the graph in DBpedia and and large dimension values of the skyline in WL II. Thus, the derived bounds used in function reverseBFS in algorithm MultiDiSkylineOpt do not eliminate many non-skyline KTrees.

Except on Yago, the average running time of MultiDiSkylineOpt under WL II is higher than that under WL I. The reason is that the skylines of the queries in WL II have large values in some dimensions, so that there are either more candidates to process or the algorithm spends substantial time on traversing long paths in the graph. On Yago, MultiDiSkylineOpt processes queries in WL II faster than those in WL I. The reason is that the keywords in Yago (WL II) are infrequent, so that the reachability testing eliminates many candidates, leaving fewer candidates behind in WL II than in WL I. On CiteSeer, the average running time of FilterRefine under WL I is lower than that under WL II. The reason is as follows. As shown in Table V, the number of distinct words in CiteSeer is smaller than in the other datasets, so that the frequencies of the words in CiteSeer are higher. Thus, it is difficult to find queries whose skylines have large values in some dimensions. As shown in Table VI, the average largest dimension value in CiteSeer (WL II) is smaller than that in other datasets. Although the average number of candidates of the queries in CiteSeer (WL II) is smaller than in CiteSeer (WL I), after reachability testing, slightly more candidates remain in WL II than in WL I. Except on CiteSeer, the average running time of FilterRefine under WL I is higher than under WL II. This is because the keyword selectivity in the queries in WL II and the reachability testing eliminate many candidates.

Overall, MultiDiSkylineOpt is the most efficient algorithm.

C. Ablation Study of the Proposed Algorithms

To understand the effectiveness of the techniques included in FilterRefine and MultiDiSkylineOpt, we conduct an ablation study by disabling techniques in the proposed algorithms. FilterRefine-P is the version without the skyline-based and inter-partition pruning. MultiDiSkyline (Algorithm 3) is the version without function reverseBFS. MultiDiSkyline-L3 is the version without Lemma 3. MultiDiSkyline-L3L2 is the version without Lemmas 2 and 3. As shown in Table IX, FilterRefine outperforms FilterRefine-P on all datasets in terms of both running time and the number of candidates, indicating that the pruning strategies are effective. The running time of MultiDiSkylineOpt is lower than that of MultiDiSkyline which means that function reverseBFS is effective at determining the distances between keywords and vertices. MultiDiSkyline is faster than MultiDiSkyline-L3 which indicates that the bounds derived in Lemma 3 help stopping the graph exploration in the BFS earlier for non-skyline KTrees. The number of candidates of MultiDiSkyline-L3L2 is much higher than for MultiDiSkyline-L3, and MultiDiSkyline-L3 is more efficient than MultiDiSkyline-L3L2, indicating that the bound derived in Lemma 2 is effective at pruning candidates.

D. Effect of Parameters

Effect of the Number of Query Keywords: Fig. 5 shows the performance of the proposed algorithms under the two types of workloads on the five datasets when varying the number of query keywords. In most cases, the average running time increases as the number of keywords increases. This is because more keywords result in more KTrees being involved in the computation. On the other hand, on Yago (WL II), CiteSeer (WL II), DBLP, and DBpedia, due to the selectivity of some keywords, the number of vertices that participate in the computation may decrease. Thus, the average running time sometimes decreases as the number of keywords increases. When the number of query keywords is 4, on Yago (WL II), DBLP (WL II), and DBpedia (WL II), FilterRefine is faster than MultiDiSkylineOpt. The reason is that the selectivity of the 4 keyword-queries may be high, so that there are relatively few candidates. In most cases, MultiDiSkylineOpt performs the best, since it is able to prune the search space effectively and computes KTrees efficiently.

TABLE VIII
AVERAGE RUNNING TIME (SECONDS) AND AVERAGE #CANDIDATES PER 3-KEYWORD QUERY

Algorithms		Datasets									
		DBpedia		Yago		DBLP		LastFm		CiteSeer	
		WL I	WL II	WL I	WL II	WL I	WL II	WL I	WL II	WL I	WL II
Baseline	running time	INF	INF	INF	INF	INF	INF	INF	INF	INF	INF
	#candidates	8,099,624.00	8,099,624.00	8,091,094.00	8,091,094.00	4,348,701.00	4,348,701.00	250,329.00	250,329.00	371,240.00	371,240.00
FilterRefine	running time	211.66	**118.10**	303.73	212.52	494.73	439.74	INF	INF	275.80	300.44
	#candidates	573,000.50	1,226,189.85	108,963.15	270,907.70	13,397.25	81,239.95	23,179.15	18,814.20	5,329.55	2,617.55
MultiDiSkylineOpt	running time	**106.74**	228.90	**161.77**	**144.82**	**127.02**	**136.03**	**72.17**	**206.31**	**82.99**	**300.36**
	#candidates	984,115.00	1,812,701.80	427,940.05	665,875.35	157,208.20	624,562.60	25,005.30	21,816.65	19,981.00	71,234.55

TABLE IX
AVERAGE RUNNING TIME (SECONDS) AND AVERAGE #CANDIDATES PER 3-KEYWORD QUERY (ABLATION STUDY)

Algorithms		Datasets									
		DBpedia		Yago		DBLP		LastFm		CiteSeer	
		WL I	WL II	WL I	WL II	WL I	WL II	WL I	WL II	WL I	WL II
FilterRefine	running time	211.66	**118.10**	303.73	212.52	494.73	439.74	INF	INF	275.80	300.44
	#candidates	573,000.50	1,226,189.85	108,963.15	270,907.70	13,397.25	81,239.95	23,179.15	18,814.20	5,329.55	2,617.55
FilterRefine-P	running time	217.80	126.27	338.20	257.55	INF	476.38	INF	INF	INF	INF
	#candidates	2,792,613.00	1,589,537.90	1,111,475.75	378,813.90	1,368,407.40	1,056,823.00	146,090.25	97,594.00	132,658.95	120,989.25
MultiDiSkylineOpt	running time	**106.74**	228.90	**161.77**	**144.82**	**127.02**	**136.03**	**72.17**	**206.31**	**82.99**	**300.36**
	#candidates	984,115.00	1,812,701.80	427,940.05	665,875.35	157,208.20	624,562.60	25,005.30	21,816.65	19,981.00	71,234.55
MultiDiSkyline	running time	244.60	263.30	337.97	297.29	448.73	473.46	467.35	448.84	378.67	451.07
	#candidates	984,115.00	1,812,701.80	427,940.05	665,875.35	157,208.20	624,562.60	25,005.30	21,816.65	19,981.00	71,234.55
MultiDiSkyline-L3	running time	247.41	268.26	339.03	308.09	491.64	481.77	475.25	INF	489.18	499.70
	#candidates	984,115.00	1,812,701.80	427,940.05	665,875.35	157,208.20	624,562.60	25,005.30	21,816.65	19,981.00	71,234.55
MultiDiSkyline-L3L2	running time	358.96	308.52	348.09	319.23	INF	495.88	INF	INF	INF	INF
	#candidates	5,132,877.15	6,449,401.05	3,021,282.65	3,553,400.75	3,791,529.00	3,809,842.10	249,906.30	250,156.50	213,993.25	229,836.75

TABLE X
STATISTICS ON THE α-HOP INVERTED INDEX

Dataset	Storage (GB)			Construction time (hours)		
	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$
DBpedia	57.90	191.00	556.00	6.70	25.00	95.00
Yago	8.93	38.80	114.00	1.10	3.20	15.30
CiteSeer	3.63	13.90	54.70	0.23	1.80	7.50

TABLE XI
PERFORMANCE OF FilterRefine WHEN VARYING α

Dataset	Average running time (seconds)					
	$\alpha = 2$		$\alpha = 3$		$\alpha = 4$	
	WL I	WL II	WL I	WL II	WL I	WL II
DBpedia	218.45	243.61	225.10	262.62	245.41	264.65
Yago	296.24	334.56	302.43	324.79	281.94	274.64
CiteSeer	259.52	331.78	241.02	284.92	192.63	186.56

Effect of the Dataset Size: To evaluate the effect of the dataset size on the performance of the proposed algorithms, we extract four subsets with different numbers of vertices and edges from Yago. On each subset, we generate 20 3-keyword queries of type WL I as described in Section V-A. Since the sizes of the subsets are not large, it is difficult to generate queries of type WL II. Fig. 6 shows the average running times of the proposed algorithms when varying the dataset size. As the numbers of vertices and edges increase, the average running time increases, which is expected. The average running time of MultiDiSkylineOpt increases more slowly than that of FilterRefine. The reason may be that the reused skylines in subspaces and the derived bounds become more helpful when the dataset size grows.

Effect of α in FilterRefine*:* Table X shows the storage costs and the construction time of the α-hop inverted index when $\alpha = 2, 3, 4$ on datasets DBpedia, Yago, and CiteSeer. Table XI shows the average running time per query of algorithm FilterRefine when using these indexes. A large α indicates that more vertices in the neighborhood of each vertex are stored. Thus, the storage cost and the construction time of the index increases as α increases, which is expected. Parameter α affects the performance of FilterRefine in two ways. First, as α increases, more candidate vertices are involved in the processing, thus increasing the running time per query. Second, as α increases, the distances from more vertices to keywords are known, thus decreasing the computational costs of graph exploration and, thereby, the running time per query. We observe these two effects on different datasets, as shown in Table XI.

E. Case Study

To illustrate the meaning of skyline KTrees, we conduct case studies on dataset CiteSeer, where vertices represent papers and edges denote references. Issuing skyline KTree queries on CiteSeer can help researchers identify relevant papers w.r.t. given topics. Fig. 7 shows two skyline KTrees of the 3-keyword query {signal processing, mean squared error, simple convex optimization problem}, where each of the skyline KTrees $\vec{v}_1$ and $\vec{v}_7$ contains three relevant papers. Fig. 8 shows two skyline KTrees of the 3-keyword query {genetic algorithm, polynomial-time algorithm, fuzzy random variable}. By reading the papers included in skyline KTrees $\vec{v}_1$ and $\vec{v}_5$, researchers may gain an understanding of relevant algorithms.

VI. RELATED WORK

Skyline Queries on Graphs: Skyline problems on graphs have been investigated with various definitions of dominance relationships. Papadopoulos et al. [23] defines a partial order on subgraphs in terms of the number of vertices and edge connectivity. The dynamic skyline query on a graph [24] considers dominance relationships between vertices based on their graph distances to the query vertices. The skyline graph pattern query on RDF

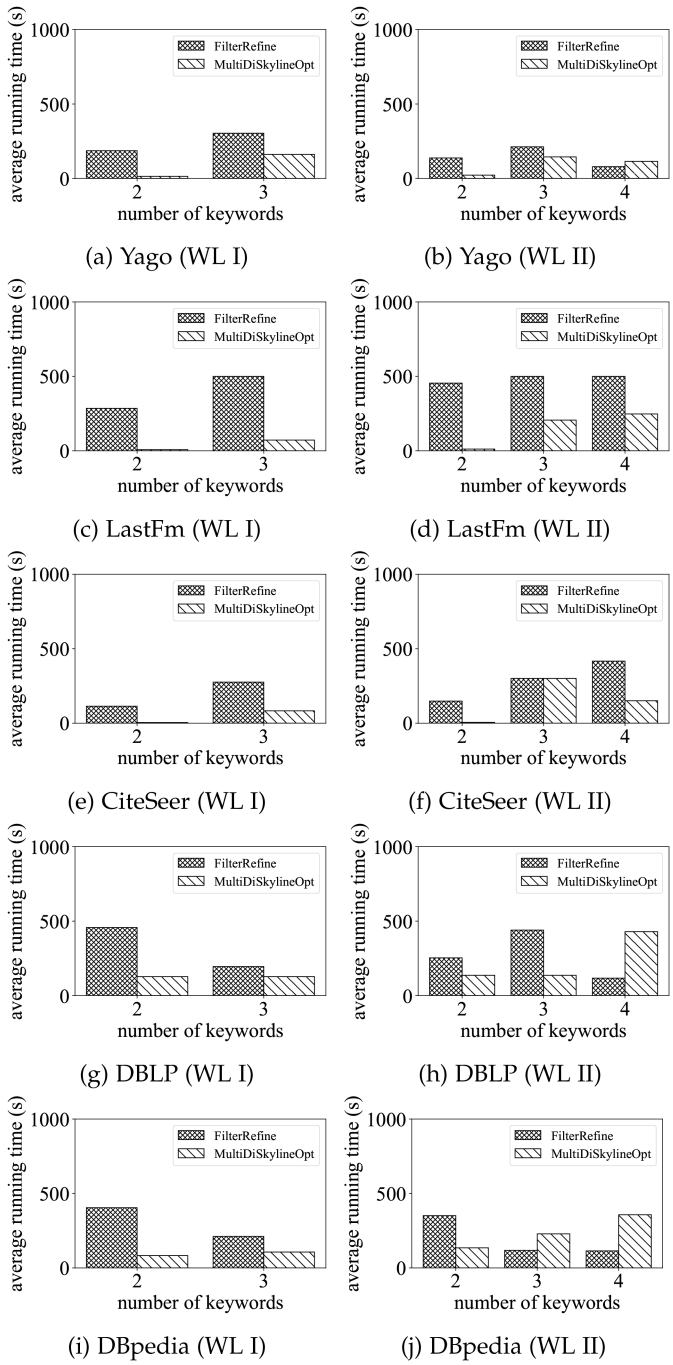


Fig. 5. Effect of the number of query keywords.

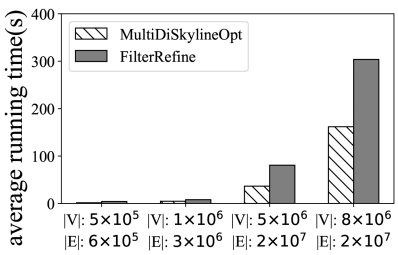


Fig. 6. Effect of the dataset size.

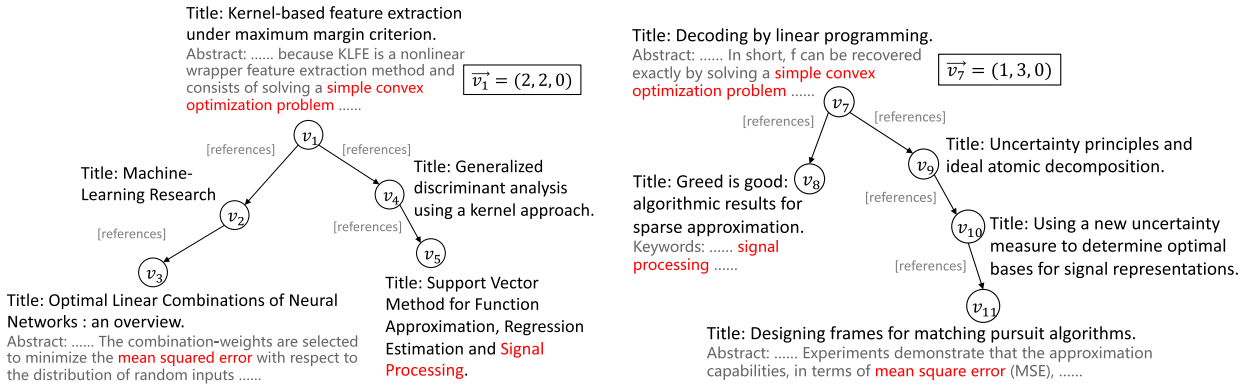
data [25] considers dominance relationships between graph patterns based on a set of pre-defined measurements. Subgraph skyline search [26], [27] returns skyline subgraphs isomorphic to the query graph pattern, where the dominance relationship between subgraphs is defined based on the numeric attributes in query. Pande et al. [28] introduce dominance relationships between subgraphs in road networks based on the number of edges and the number of query keywords covered. SkyTPF [29] enables skyline query processing over standard RDF interfaces and returns skyline results of graph pattern queries. Skyline community search [30] finds communities that are not dominated by other communities in terms of d numerical attributes in a multi-valued network. A skyline community is defined as a maximal connected k-core that cannot be dominated by other connected k-cores in the d-dimensional attribute space. The skyline cohesive group query [31] on road-social networks finds groups of users that cannot be dominated by any other group in terms of social cohesiveness and spatial cohesiveness. Dhifli et al. [32] introduce dominance relationships between clustering solutions on attributed graph data based on a set of predefined quality measures. Skyline path queries [33], [34] extend skyline queries to multi-cost road networks and return sets of non-dominated paths between given pairs of network nodes. The dominance relationship between paths is determined by multiple costs, such as distance, travel time, the number of traffic lights, and fuel consumption.

The definitions of the dominance relationships and the skyline results in these problems are different from the setting we consider. Each of the proposed techniques is dedicated for a specific problem and cannot be applied directly to solve our problem.

Keyword Queries on Graphs: A keyword query on graphs returns top-k most relevant substructures. According to ranking models, relevant substructures can determined according to tree-based semantics, subgraph-based semantics, nearest neighbor-based semantics, and other semantics [4], [5]. In tree-based query semantics, an answer is a tree extracted from the graph that covers all the query keywords, and answers are ranked by Steiner tree-based costs [6], [35], [36], [37], distinct root-based costs [3], [38], [39], [40], or IR-style weighing [41], [42], [43]. The answers in subgraph-based query semantics are subgraphs that cover all query keywords, and they are ranked according to Steiner graph-based costs [7], [44], community-based costs [1], [8], [45], [46], [47], or language model based scoring [48]. The nearest neighbor-based semantics retrieve vertices in the graph that are both relevant to the query keywords and close to a query vertex in terms of path length [2], [49], [50].

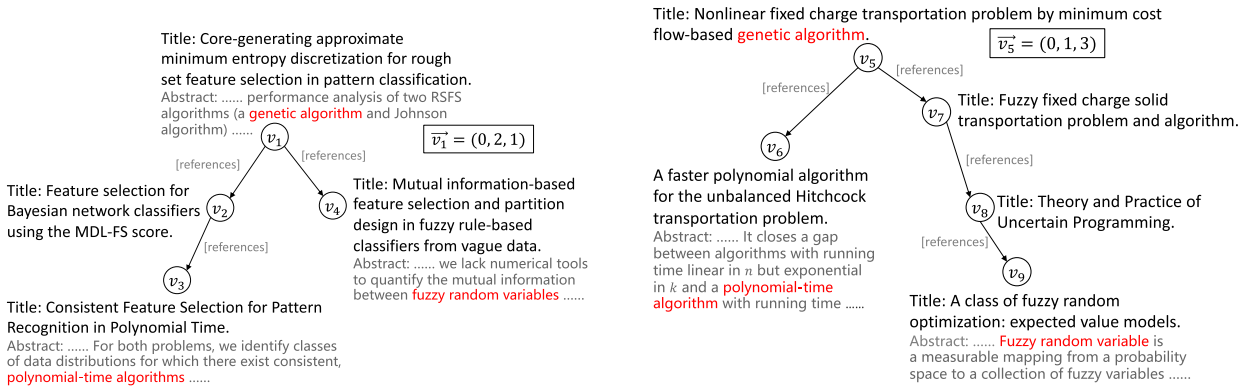
This paper investigates the distinct root-based costs in tree-based semantics and adopts the notion of skyline to eliminate the strong reliance on a single ranking function.

Skyline Computation: The skyline operator [12] identifies a set of interesting points in a potentially large set of data points based on so-called dominance relationships. Early solutions, such as the divide-and-conquer (D&C) approach and the block-nested-loop (BNL) algorithm [12] do not scale well on large datasets. The performance of the D&C approach deteriorates in



(a) $\vec{v}_1 = (2, 2, 0)$. (b) $\vec{v}_7 = (1, 3, 0)$.

Fig. 7. Skyline KTrees of keywords {signal processing, mean squared error, simple convex optimization problem}.



(a) $\vec{v}_1 = (0, 2, 1)$. (b) $\vec{v}_5 = (0, 1, 3)$.

Fig. 8. Skyline KTrees of keywords {genetic algorithm, polynomial-time algorithm, fuzzy random variable}.

high dimensional spaces, and the BNL algorithm requires a large number of passes on the dataset to obtain results. The sort-filter-skyline (SFS) [51] algorithm improves on BNL by processing points in topological order based on a monotone function, such that the points following point p cannot dominate p. The linear-elimination-sort-for-skyline (LESS) [52] algorithm optimizes SFS by obtaining a small set of objects in the initial pass, such that other points dominated by these initial points can be pruned early. The SaLSa [22] algorithm sorts points according to their minimum coordinate value in all dimensions and identifies $stop$ objects in $O(1)$ time, so that remaining comparisons can be avoided. The distribution-sweep [53] algorithm solves the skyline merge problem in external memory. Lee et al. [54] develop a pivot selection technique that balances the dominance and the incomparability of points. They propose algorithm BSkyTree that embeds the pivot point selection technique into existing skyline algorithms.

A branch of approaches adopt indexes for skyline computations. Tan et al. [55] return skyline points progressively. All points are transformed into a single-dimensional space, are partitioned, and are stored in B^+-tree structures. Kossmann et al. [56] adopt an R-tree to return the nearest neighbor to the origin as a skyline point. All remaining points are partitioned based on the skyline point. Other skyline points are found iteratively in each partition using the R-tree. BBS [57] adopts the branch-and-bound paradigm on an R-tree indexing all points. Skyline points are retrieved progressively from unpruned R-tree nodes. Lee et al. [58] index all the points in the ZBtree based on the Z-order curve and compute skyline points progressively using an algorithm called ZSearch. To reduce the numerous computations required for comparing objects in high dimensional spaces, Zhang et al. [59] propose a dynamic index for skyline points that can be integrated into state-of-the-art sort-based skyline algorithms. SSPL [60] constructs a sorted positional index list for each dimension, where points are arranged in ascending order. It first finds an object that can dominate all objects that have not been visited. Then, an existing sorting-based skyline algorithm is applied to compute skyline results. To reduce the cost of point comparisons, the MBR-oriented approach [61] adopts an R-tree to perform dominance testing among skylines of minimum bounding rectangles. Scalable Dimension Indexing (SDI) [62] tests a point only with the existing skyline points on an arbitrary dimension instead of with the complete set of skyline points. The cost of traversal on the indexed dimensions is less than that of SaLSa. Recently, Miao et al. [63] study skyline queries over incomplete data with crowdsourcing.

Further, EECE [64] is shown to significantly accelerate cardinality estimation for the skyline query family.

All the above techniques are designed based on the assumption that the values in all dimensions are known. However, in our problem, it is excessively costly to materialize all dimensions of points.

The skyline operator has recently been extended to various scenarios to discover interesting results. Skyline Diagram [65] partitions the plane into regions, so that all query points in the same region have the same skyline query results. This can be used to facilitate skyline queries. G-Skyline [66], [67] generalizes the original skyline definition to find Pareto optimal groups of points that are not dominated by other groups. Secure dynamic skyline queries [68] evaluate skylines on encrypted datasets. Probabilistic skyline computation [69] on incomplete data returns k points with the highest skyline probabilities. The F-skyline operator [70] defines dominance with respect to a family of scoring functions. Our study is orthogonal to these studies and adds new functionality to skyline research.

Multidimensional Subspace Skyline Computation: To compute multidimensional subspace skylines, the state-of-the-art skycube [17] materializes the skylines of all possible nonempty subspaces of a given full space. Subsequent studies improve on skycube by reducing its storage cost or by speeding up its computations. Raïssi et al. [71] propose a concise skycube, which reduces the number of subspaces that need to be searched and the domination tests in a given subspace. CSC [72] reduces the storage cost of skycubes by storing minimum skylines of subspaces. QSkycube [13] optimizes the computation of skycubes by using point-based space partitioning. Hashcube [73] compresses skycubes using a hashing- and bitstring-based data structure. Maabout et al. [14] find that materializing the topmost skyline (the skyline on all dimensions) is sufficient for multidimensional skyline query optimization. RSkycube [15] speeds up skycube computation by sharing the computations among multiple related cuboids. The negative skycube [16] encodes subspaces with respect to which tuple t is dominated.

In our problem, if considering all the keywords in the data as the full space, the query keywords can be regarded as a subspace. However, these algorithms are applicable only when the values of all dimensions are available before querying starts. In our setting, considering the high cost of materializing all dimensions, we index skyline results in single-dimensional space and propose algorithms for efficient skyline computation in multi-dimensional space.

VII. Conclusion

We define the problem of skyline KTree retrieval. To enable efficient computation of skyline KTrees on large attributed graphs, we propose a filter-and-refinement algorithm and subspace based algorithms. An extensive experimental study shows that the filter-and-refinement algorithm is efficient when keyword distance computation is needed for only relatively few candidates. The subspace based algorithms are capable of efficiently computing skyline results in most cases. In future research, it is of interest to consider the the problem of skyline KTree retrieval in distributed computing environments. Moreover, to reduce the number of skyline KTrees in results, developing top-k, constrained, interactive, and weighted variants of the skyline KTree retrieval query is of interest.

References

[1] M. S. Islam, M. E. Ali, Y. Kang, T. Sellis, F. M. Choudhury, and S. Roy, "Keyword aware influential community search in large attributed graphs," *Inf. Syst.*, vol. 104, pp. 1–15, 2022.

[2] T. Abeywickrama, M. A. Cheema, and A. Khan, "K-SPIN: Efficiently processing spatial keyword queries on road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 5, pp. 983–997, May 2020.

[3] W. Le, F. Li, A. Kementsietsidis, and S. Duan, "Scalable keyword search on large RDF data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 11, pp. 2774–2788, Nov. 2014.

[4] J. Yang, W. Yao, and W. Zhang, "Keyword search on large graphs: A survey," *Data Sci. Eng.*, vol. 6, no. 2, pp. 142–162, 2021.

[5] Y. Wang, Y. Li, J. Fan, C. Ye, and M. Chai, "A survey of typical attributed graph queries," *World Wide Web*, vol. 24, no. 1, pp. 297–346, 2021.

[6] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *Proc. IEEE Int. Conf. Data Eng.*, 2007, pp. 836–845.

[7] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou, "EASE: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 903–914.

[8] Y. Fang, Y. Yang, W. Zhang, X. Lin, and X. Cao, "Effective and efficient community search over large heterogeneous information networks," in *Proc. VLDB Endowment*, vol. 13, no. 6, pp. 854–867, 2020.

[9] Y. Shi, G. Cheng, T. Tran, E. Kharlamov, and Y. Shen, "Efficient computation of semantically cohesive subgraphs for keyword-based knowledge graph exploration," in *Proc. Web Conf.*, 2021, pp. 1410–1421.

[10] D. Wu, H. Zhou, J. Shi, and N. Mamoulis, "Top-k relevant semantic place retrieval on spatiotemporal RDF data," *VLDB J.*, vol. 29, no. 4, pp. 893–917, 2020.

[11] S. E. Dreyfus and R. A. Wagner, "The Steiner problem in graphs," *Networks*, vol. 1, no. 3, pp. 195–207, 1971.

[12] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. IEEE Int. Conf. Data Eng.*, 2001, pp. 421–430.

[13] J. Lee and S. Hwang, "Toward efficient multidimensional subspace skyline computation," *VLDB J.*, vol. 23, no. 1, pp. 129–145, 2014.

[14] S. Maabout, C. Ordonez, P. K. Wanko, and N. Hanusse, "Skycube materialization using the topmost skyline or functional dependencies," *ACM Trans. Database Syst.*, vol. 41, no. 4, pp. 25:1–25:40, 2016.

[15] K. Zhang, H. Gao, X. Han, D. Yang, Z. Cai, and J. Li, "Rskycube: Efficient skycube computation by reusing principle," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2017, pp. 51–64.

[16] K. Alami, N. Hanusse, P. K. Wanko, and S. Maabout, "The negative skycube," *Inf. Syst.*, vol. 88, pp. 1–17, 2020.

[17] J. Pei et al., "Towards multidimensional subspace skyline analysis," *ACM Trans. Database Syst.*, vol. 31, no. 4, pp. 1335–1381, 2006.

[18] J. Cheng, S. Huang, H. Wu, and A. W. Fu, "TF-label: A topological-folding labeling scheme for reachability querying in a large graph," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 193–204.

[19] A. V. Goldberg, H. Kaplan, and R. F. Werneck, "Reach for A*: Efficient point-to-point shortest path algorithms," in *Proc. Workshop Algorithm Eng. Experiments*, SIAM, 2006, pp. 129–143.

[20] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "ArnetMiner: Extraction and mining of academic social networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2008, pp. 990–998.

[21] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proc. 12th Int. Soc. Music Inf. Retrieval Conf.*, 2011, pp. 591–596.

[22] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient sort-based skyline evaluation," *ACM Trans. Database Syst.*, vol. 33, no. 4, pp. 31:1–31:49, 2008.

[23] A. N. Papadopoulos, A. Lyritsis, and Y. Manolopoulos, "SkyGraph: An algorithm for important subgraph discovery in relational graphs," *Data Mining Knowl. Discov.*, vol. 17, no. 1, pp. 57–76, 2008.

[24] L. Zou, L. Chen, M. T. Özsu, and D. Zhao, "Dynamic skyline queries in large graphs," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2010, pp. 62–78.

[25] L. Chen, S. Gao, and K. Anyanwu, "Efficiently evaluating skyline queries on RDF databases," in *Proc. Extended Semantic Web Conf.*, 2011, pp. 123–138.

[26] W. Zheng, L. Zou, X. Lian, L. Hong, and D. Zhao, "Efficient subgraph skyline search over large graphs," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2014, pp. 1529–1538.
[27] W. Zheng, X. Lian, L. Zou, L. Hong, and D. Zhao, "Online subgraph skyline analysis over knowledge graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1805–1819, Jul. 2016.
[28] S. Pande, S. Ranu, and A. Bhattacharya, "SkyGraph: Retrieving regions of interest using skyline subgraph queries," in *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1382–1393, 2017.
[29] I. Keles and K. Hose, "Skyline queries over knowledge graphs," in *Proc. Int. Semantic Web Conf.*, 2019, pp. 293–310.
[30] R. Li et al., "Finding skyline communities in multi-valued networks," *VLDB J.*, vol. 29, no. 6, pp. 1407–1432, 2020.
[31] Q. Li, Y. Zhu, and J. X. Yu, "Skyline cohesive group queries in large road-social networks," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 397–408.
[32] W. Dhifli, N. E. I. Karabadji, and M. Elati, "Evolutionary mining of skyline clusters of attributed graph data," *Inf. Sci.*, vol. 509, pp. 501–514, 2020.
[33] Z. Liu, L. Li, M. Zhang, W. Hua, and X. Zhou, "Multi-constraint shortest path using forest hop labeling," *VLDB J.*, vol. 32, pp. 595–621, 2023.
[34] Q. Gong and H. Cao, "Backbone index to support skyline path queries over multi-cost road networks," in *Proc. 25th Int. Conf. Extending Database Technol.*, 2022, pp. 2:325–2:337.
[35] R. Li, L. Qin, J. X. Yu, and R. Mao, "Efficient and progressive group steiner tree search," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 91–106.
[36] Y. Shi, G. Cheng, and E. Kharlamov, "Keyword search over knowledge graphs via static and dynamic hub labelings," in *Proc. Web Conf.*, 2020, pp. 235–245.
[37] Y. Shi, G. Cheng, T. Tran, J. Tang, and E. Kharlamov, "Keyword-based knowledge graph exploration based on quadratic group steiner trees," in *Proc. Int. Joint Conf. Artif. Intell.*, 2021, pp. 1555–1562.
[38] H. He, H. Wang, J. Yang, and P. S. Yu, "BLINKS: Ranked keyword searches on graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 305–316.
[39] Y. Luo, W. Wang, X. Lin, X. Zhou, J. Wang, and K. Li, "SPARK2: Top-k keyword query in relational databases," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 12, pp. 1763–1780, Dec. 2011.
[40] Z. Zhang, J. X. Yu, G. Wang, Y. Yuan, and L. Chen, "Key-core: Cohesive keyword subgraph exploration in large graphs," *World Wide Web*, vol. 25, no. 2, pp. 831–856, 2022.
[41] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-style keyword search over relational databases," in *Proc. 29th Int. Conf. Very Large Data Bases*, 2003, pp. 850–861.
[42] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury, "Effective keyword search in relational databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 563–574.
[43] Y. Xu, J. Guan, F. Li, and S. Zhou, "Scalable continual top-k keyword search in relational databases," *Data Knowl. Eng.*, vol. 86, pp. 206–223, 2013.
[44] M. Kargar, L. Golab, D. Srivastava, J. Szlichta, and M. Zihayat, "Effective keyword search over weighted graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 2, pp. 601–616, Feb. 2022.
[45] Q. Liu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Truss-based community search over large drected graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 2183–2197.
[46] Q. Liu, Y. Zhu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "VAC: Vertex-centric attributed community search," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 937–948.
[47] Y. Zhu, Q. Zhang, L. Qin, L. Chang, and J. X. Yu, "Cohesive subgraph search using keywords in large networks," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 178–191, Jan. 2022.
[48] A. Ghanbarpour and H. Naderi, "An attribute-specific ranking method based on language models for keyword search over graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 1, pp. 12–25, Jan. 2020.
[49] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *VLDB J.*, vol. 21, no. 6, pp. 797–822, 2012.
[50] R. Zhong, G. Li, K. Tan, L. Zhou, and Z. Gong, "G-tree: An efficient and scalable index for spatial search on road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 8, pp. 2175–2189, Aug. 2015.
[51] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proc. IEEE Int. Conf. Data Eng.*, 2003, pp. 717–719.
[52] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," in *Proc. Int. Conf. Very Large Data Bases*, 2005, pp. 229–240.
[53] C. Sheng and Y. Tao, "Worst-case I/O-efficient skyline algorithms," *ACM Trans. Database Syst.*, vol. 37, no. 4, pp. 26:1–26:22, 2012.
[54] J. Lee and S. Hwang, "Scalable skyline computation using a balanced pivot selection technique," *Inf. Syst.*, vol. 39, pp. 1–21, 2014.
[55] K. Tan, P. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proc. Int. Conf. Very Large Data Bases*, 2001, pp. 301–310.
[56] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *Proc. Int. Conf. Very Large Data Bases*, 2002, pp. 275–286.
[57] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.
[58] K. C. K. Lee, B. Zheng, H. Li, and W. Lee, "Approaching the skyline in Z order," in *Proc. Int. Conf. Very Large Data Bases*, 2007, pp. 279–290.
[59] S. Zhang, N. Mamoulis, and D. W. Cheung, "Scalable skyline computation using object-based space partitioning," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 483–494.
[60] X. Han, J. Li, D. Yang, and J. Wang, "Efficient skyline computation on big data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 11, pp. 2521–2535, Nov. 2013.
[61] J. Zhang, W. Wang, X. Jiang, W. Ku, and H. Lu, "An MBR-oriented approach for efficient skyline query processing," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 806–817.
[62] R. Liu and D. Li, "Efficient skyline computation in high-dimensionality domains," in *Proc. 23rd Int. Conf. Extending Database Technol.*, 2020, pp. 459–462.
[63] X. Miao, Y. Gao, S. Guo, L. Chen, J. Yin, and Q. Li, "Answering skyline queries over incomplete data with crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1360–1374, Apr. 2021.
[64] X. Miao, Y. Wu, J. Peng, Y. Gao, and J. Yin, "Efficient and effective cardinality estimation for skyline family," in *Proc. ACM Manag. Data*, vol. 1, no. 1, pp. 104:1–104:21, 2023.
[65] J. Liu et al., "Skyline diagram: Efficient space partitioning for skyline queries," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 1, pp. 271–286, Jan. 2021.
[66] J. Liu, L. Xiong, J. Pei, J. Luo, H. Zhang, and W. Yu, "Group-based skyline for pareto optimal groups," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 7, pp. 2914–2929, Jul. 2021.
[67] W. Yu, J. Liu, J. Pei, L. Xiong, X. Chen, and Z. Qin, "Efficient contour computation of group-based skyline," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 7, pp. 1317–1332, Jul. 2020.
[68] S. Zeighami, G. Ghinita, and C. Shahabi, "Secure dynamic skyline queries using result materialization," in *Proc. IEEE Int. Conf. Data Eng.*, 2021, pp. 157–168.
[69] K. Zhang, H. Gao, X. Han, Z. Cai, and J. Li, "Modeling and computing probabilistic skyline on incomplete data," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 7, pp. 1405–1418, Jul. 2020.
[70] P. Ciaccia and D. Martinenghi, "Flexible skylines: Dominance for arbitrary sets of monotone functions," *ACM Trans. Database Syst.*, vol. 45, no. 4, pp. 18:1–18:45, 2020.
[71] C. Raïssi, J. Pei, and T. Kister, "Computing closed skycubes," in *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 838–847, 2010.
[72] T. Xia, D. Zhang, Z. Fang, C. X. Chen, and J. Wang, "Online subspace skyline query processing using the compressed skycube," *ACM Trans. Database Syst.*, vol. 37, no. 2, pp. 15:1–15:36, 2012.
[73] K. S. Bøgh, S. Chester, D. Sidlauskas, and I. Assent, "Hashcube: A data structure for space- and query-efficient skycube compression," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2014, pp. 1767–1770.

Dingming Wu received the bachelor's degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2005, the master's degree in computer science from Peking University, Beijing, China, in 2008, and the PhD degree in computer science from Aalborg University, Denmark, in 2011. She is associate professor with the College of Computer Science and Software Engineering, Shenzhen University, China. Her research concerns data management and analytics, query processing, and data mining.

Zhaofen Zhang received the bachelor's degree from the Dongguan University of Technology, China, in 2020. He is currently working toward the master's degree with the College of Computer Science and Software Engineering, Shenzhen University, China. His research interests include data management and query processing.

Christian S. Jensen (Fellow, IEEE) is professor of computer science with Aalborg University, Denmark. His research concerns analytics, data management, and data-intensive systems, and it centers around temporal and spatio-temporal analytics, including machine learning, data mining, and query processing. He is a fellow of the ACM, and he is a member of Academia Europaea, the Royal Danish Academy of Sciences and Letters, and the Danish Academy of Technical Sciences. He has received several awards, most recently the 2022 ACM SIGMOD Contributions Award.

Kezhong Lu received the bachelor's and PhD degrees in computer science from the University of Science and Technology of China, in 2001 and 2006, respectively. He is professor with the College of Computer Science and Software Engineering, Shenzhen University, China. His research concerns Big Data, parallel and distributed computing, algorithms, wireless sensor network, and computational geometry. He is a vice dean with the College of Computer Science & Software Engineering and the leader of computer technology area with Shenzhen University, China.