This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

Online Subgraph Skyline Analysis Over Knowledge Graphs

Weiguo Zheng[1,4], Xiang Lian[2], Lei Zou[1], Liang Hong[3], Dongyan Zhao[1]

[1] *Peking University, Beijing, China, 100080;*

[2] *University of Texas Rio Grande Valley, Edinburg, Texas, USA, 78539;*

[3] *Wuhan University, Wuhan, China, 430072;*

[4] *The Chinese University of Hong Kong, Hong Kong, China, 999077.*

{zhengweiguo,zoulei,zhaody}@pku.edu.cn, xiang.lian@utrgv.edu

Abstract—Subgraph search is very useful in many real-world applications. However, users may be overwhelmed by the masses of matches. In this paper, we propose a subgraph skyline analysis problem, denoted as S^2A, to support more complicated analysis over graph data. Specifically, given a large graph G and a query graph q, we want to find all the subgraphs g in G, such that g is graph isomorphic to q and not dominated by any other subgraphs. In order to improve the efficiency, we devise a hybrid feature encoding incorporating both structural and numeric features based on a partitioning strategy, and discuss how to optimize the space partitioning. We also present a skylayer index to facilitate the dynamic subgraph skyline computation. Moreover, an attribute cluster-based method is proposed to deal with the curse of dimensionality. Extensive experiments over real datasets confirm the effectiveness and efficiency of our algorithm.

Index Terms—Subgraph Skyline; Feature Encoding; Skylayer; High Dimensionality

1 Introduction

Recently, graphs have attracted the increasing attention from the database community [1]. A lot of real-world data (e.g., social networks [2], knowledge graphs [3], heterogenous information networks [4], and the Semantic Web [5]) can be represented by the graph model. In the literature, various research problems over graphs have been investigated, such as the shortest path query [6], subgraph search [1], [7], the reachability query [8], and so on.

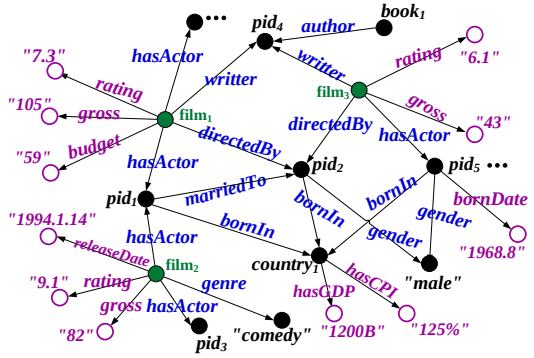


Fig. 1. An example of knowledge graph.

As a well-known research problem, the subgraph search problem is meaningful and useful in many applications. For example, answering SPARQL queries in the Semantic Web is equivalent to conducting the subgraph match over graphs [5]. However, users may be overwhelmed by the enormous matching results returned from queries. Owing to different requirements in a variety of applications, it is non-trivial how to design a generic function to measure/rank the "goodness" of these matches. In this paper, we study the subgraph skyline problem (Def. 2.5) on large graphs, which can retrieve the matching subgraphs by considering both graph structures and graph entity dominance relationships.

We first provide two motivation examples for the subgraph skyline problem below.

1.1 Motivating Example

Motivating Example 1. We want to find excellent NBA player partners in the same team over the knowledge graph as shown in Fig. 1. Specifically, one player is a "guard" with excellent techniques in "assists" and "steals". The other player is a "forward" with excellent techniques in "rebounds" and "blocks". This query can be represented by a graph in Fig. 2, where $player_1$ is a "guard" and $player_2$ is a "forward". The vertices labeled with ' * ' are their respective numeric attributes of technical statistics. In this example, we want to find all the partners serving in the same teams, who are not worse than other partners in terms of these attributes. This type of query over knowledge

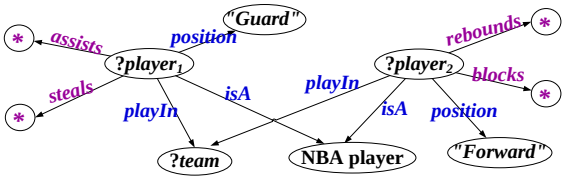


Fig. 2. Excellent basketball partner analysis.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

graphs exactly corresponds to a subgraph skyline query, and can find excellent NBA player partners.

Motivating Example 2. We can also explore excellent actors/actresses in the USA who are singers as well. Specifically, the gross of the film that the actor/actress starred in and the number of his/her album copies are expected to be large. Fig. 3 illustrates this query graph. In general, the subgraph search query may return too many matches if we do not consider numeric attributes in graph entities (e.g., actor/actress). Hence, the subgraph skyline analysis is useful over the knowledge graph to return American actors/actresses/singers who have both high film gross and a large number of album copies.

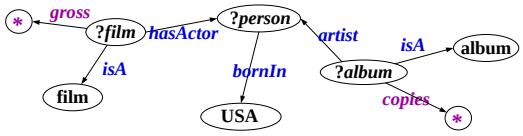


Fig. 3. Versatile artist analysis.

Motivated by the examples above, we propose the problem of **S**ubgraph **S**kyline **A**nalysis over large graphs (denoted as S^2A). Specifically, given a large graph G and a user-specified query graph pattern q (which contains numeric attributes in graph entities), S^2A returns all the subgraphs in G that are isomorphic to q and not dominated by any other isomorphic subgraphs in terms of numeric attributes in q (formally defined in Def. 2.5).

1.2 Challenges and Contributions

We address two major challenges to conduct S^2A efficiently, and carefully design the corresponding solutions.

Challenge 1: Expensive structure checking. To answer S^2A, we need to check the structural constraint (i.e., isomorphism checking) before reporting true S^2A answers. For instance, an entity v cannot be in the answers if the subgraphs containing v are not isomorphic to the query graph q, even though v is in the skyline without considering the structure constraint. However, it is NP-hard to check the graph isomorphism [9]. In order to improve the time efficiency, we should reduce the search space and avoid as many costly subgraph isomorphism checkings as possible. Thus, it is better to consider the structural feature, as well as the numeric feature. Since there may be a mass of numeric features (e.g., the numeric attributes) and structural features (e.g., path, tree, and subgraph) especially when the knowledge graph G is very large, we should carefully select these features to enhance the pruning power, and organize them in an efficient way so as to reduce the storage cost.

Challenge 2: Curse of dimensionality. In order to improve the computing efficiency of skyline entities, it is required to devise an efficient index for these numeric attributes. However, there may be a large number of numeric attributes (i.e., high dimensionality) in knowledge graphs (e.g., DBpedia has 870 numeric attributes), which indicates that the dominating relationships are very rare. Hence, the computing efficiency and pruning ability degrade accordingly. This phenomenon is named as "curse of dimensionality" [10].

TABLE 1
Frequently-used Notations

Notation	Definition and Description
G	the knowledge graph
q	the query graph
D	$D = \{d_1, \ldots, d_{\|D\|}\}$, the numeric attribute space
u, v	the vertices in q and G, respectively
n	the number of numeric vertices in G
B	a grid cell in the multi-dimensional space
c	the minimal corner of B
K	the number of grid cells
$lbStr(v)$	the local structure encoding of v
$lbStr(B)$	the local structure encoding of B
$gbStr(v)$	the global structure encoding of v
$nbStr(B)$	the numeric encoding of B
$nbStr(v)$	the numeric encoding of v
$M(v)$	the vertices dominated by v

In order to tackle these challenges, we partition the data space into grid cells so that we can compute skylines cell by cell instead of entity by entity. We also carefully devise a hybrid encoding incorporating both structural and numeric features at low storage cost. To achieve better pruning effectiveness, we propose optimizations on how to find a good partitioning. Furthermore, we maintain the grid cells using an efficient index to facilitate the dynamic computation of skyline entities. More importantly, we prune the unpromising cells that cannot generate true S^2A answers by exploiting the encoding and partitioning strategies. In order to deal with the curse of dimensionality, we propose to cluster numeric attributes, and provide an efficient algorithm to compute skylines over clusters of attributes.

In summary, we make the following contributions.

- We propose the problem of **s**ubgraph **s**kyline **a**nalysis (denoted by S^2A) over large graphs, and present an efficient method to answer S^2A queries.
- We partition the data space into grid cells, and compute skylines cell by cell, instead of entity by entity. Most importantly, we propose optimizations on how to find a good partitioning.
- We propose a hybrid feature encoding incorporating both structural and numeric features to enhance the pruning ability. We also maintain the grid cells using an efficient index in order to facilitate the dynamic computation of subgraph skyline.
- By clustering numeric attributes we devise an effective method to deal with the curse of dimensionality.
- Extensive experiments over real dataset have demonstrated the effectiveness and efficiency of our method.

2 Subgraph Skyline

We formally define the subgraph skyline problem. Table 1 lists the frequently-used notations in this paper.

2.1 The Subgraph Skyline Analysis Problem

Definition 2.1: (**Knowledge Graph**). A knowledge graph is defined as $G = (V, E, L)$, where each vertex $v \in V$ represents an entity or a numeric value, each $e = (v_i, v_j) \in E$ represents a directed edge from vertex v_i to vertex v_j, and $L(v)$ (resp. $L(e)$) is the label of vertex v (resp. edge e).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

Fig. 1 shows an example of knowledge graph. If entity v has some numeric attributes, v is called a *numeric entity*. Let $v.d$ denote the value on numeric attribute d of v.

Definition 2.2: (**Graph Isomorphism**). Given two subgraphs g_1 and g_2 in graph G, g_1 is graph isomorphic to g_2 *iff* there exists a bijective function $f(.)$ such that (1) for each vertex $v \in g_1$ (excluding numeric values), $f(v) \in g_2 \wedge L(v) = L(f(v))$; (2) for each $e = (v_i, v_j) \in g_1$, we have $f(e) = (f(v_i), f(v_j)) \in g_2$, and $L(e) = L(f(e))$.

Definition 2.3: (**Dominant/Equivalent Entity**). Given two numeric entities v_1 and v_2 in a knowledge graph G and their numeric attribute set D, v_1 *dominates* v_2, denoted by $v_1 \prec v_2$, if (1) for each numeric attribute d_i, $v_1.d_i \leq v_2.d_i$, and (2) there exists at least one attribute d_j such that $v_1.d_j < v_2.d_j$. We say v_1 is *equivalent* to v_2, denoted by $v_1 = v_2$, if $v_1.d_i = v_2.d_i$ on each numeric attribute $d_i \in D$.

To facilitate the presentation, let $v_1 \preceq v_2$ denote that entity v_1 dominates or is equivalent to entity v_2.

Note that, a numeric entity v may not contain all the $|D|$ attribute values, i.e., v has missing values on some dimensions. We can utilize the method in [11] to convert an incomplete entity v to a complete one v'. It estimates a concrete value for each missing value by using a probabilistic distribution function formed by the non-missing values on the relevant dimension d_i.

Definition 2.4: (**Subgraph Dominating Relationship**). Given two subgraphs g_1 and g_2 in G, g_1 *dominates* g_2, if

- g_1 is graph isomorphic to g_2 without considering numeric values;
- It holds that $v_i \preceq f(v_i)$ for each numeric entity $v_i \in g_1$;
- There is at least a numeric entity $v_j \in g_1$ satisfying that $v_j \prec f(v_j)$.

where $f(\cdot)$ is the mapping function defined in Def. 2.2. Correspondingly, we also say g_2 is dominated by g_1.

Definition 2.5: (**Subgraph Skyline**). A subgraph $g \in G$ is in the *subgraph skyline,* if g is graph isomorphic to the query graph q (without considering the numeric attributes and values) and not dominated by any other subgraphs $g' \in G$, on those specified numeric attributes in q.

Subgraph Skyline Analysis (denoted as S^2A). Given a graph G and a query graph q containing numeric attributes, the S^2A problem is to compute subgraph skylines on G.

Complexity Analysis. It is intuitive that the S^2A problem has high complexity due to the constraints of both subgraph isomorphism and dominance relationships. If the S^2A problem does not have the numeric constraint, it is equivalent to a subgraph search problem. Since the subgraph isomorphism is a classical NP-hard problem, S^2A is also NP-hard and intractable from the perspective of the time complexity.

2.2 Straightforward Methods

Naive method 1 - NaiveIso. High-level idea: In the offline phase, we store the bitmaps of numeric entities. In the online phase, we enumerate all subgraphs that are graph isomorphic to the query. Then, we compute skyline entities by employing previously maintained bitmaps to obtain the final answers. The framework is outlined in Alg. 1.

To make it self-contained, we briefly review the bitmap method [12]. Its main idea is to represent an object $o = \{o_1, \ldots, o_{|D|}\}$ using an m-bit vector, where o_i is the value on dimension i, and o_i is represented by k_i bits. Since k_i is n at most, m is $n \cdot |D|$ in the worst case, where n and $|D|$ are the numbers of numeric entities and numeric attributes, respectively. Then we can progressively determine whether o is in the skyline by performing bitwise operations over the corresponding bitmaps.

Obviously, this naive method is inefficient, since it invokes expensive subgraph isomorphism matching to enumerate all subgraph candidates. Furthermore, its storage cost is $O(n^2 \cdot |D|)$, because there are n numeric entities, and each entity o is represented by an $(n \cdot |D|)$-bit vector in the worst case as discussed above.

Naive method 2 - NaiveJoin. The main idea is that we first utilize the skyline-join pruning techniques (i.e., filtering those entities that are dominated by others) to obtain skyline entities. Then, we verify the structural constraint by performing the subgraph isomorphism checking. Obviously, this method may generate too many skyline entities, which incurs high cost of checking graph isomorphism.

3 Hybrid Feature Encoding

Below, we propose to partition the data space into grid cells in Section 3.1, and encode for each grid cell both structural and numeric features in Section 3.2.

3.1 The Space Partitioning

The rationale of partitioning the data space (i.e., the space consisting of numeric entities) is that: if we can compute the skyline entities cell by cell instead of exhaustively searching them one by one, then a lot of search cost can be saved. Moreover, we maintain only encodings for cells rather than entities, which greatly reduces the storage cost. Thus, we propose to partition the data space into grid cells (Def. 3.1).

Definition 3.1: **(Grid).** Given a data space with numeric attributes $D = \{d_1, \ldots, d_{|D|}\}$, grids are obtained by partitioning each dimension $d_i \in D$ using a set of partitioning lines $p_i^1, \ldots, p_i^k$ ($p_i^1 < \ldots < p_i^k$). A **cell**, denoted by B, is the block bounded by two partitioning lines p_i^j and p_i^{j+1} on each dimension d_i, i.e., $p_i^j < B.d_i \leq p_i^{j+1}$

Definition 3.2: (**Minimal Corner**). Given a grid cell B in the multi-dimensional data space, its minimal corner is the point $c \in B$ whose value on each dimension $d_i \in D$ is the minimum, i.e., $c.d_i = p_i^j$.

Algorithm 1 NaiveIso

Input: A knowledge graph G; A query graph q;
Output: Subgraph skylines $\mathbb{A}$ over G.

```
1: Build the bitmaps for numeric entities
2: 𝔸 ← ∅
3: S ← subgraphs in G that are graph isomorphic to q
4: for each candidate graph g ∈ S do
5:     if the numeric entities in g belong to skyline then
6:         add g into 𝔸
7: return 𝔸
```

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

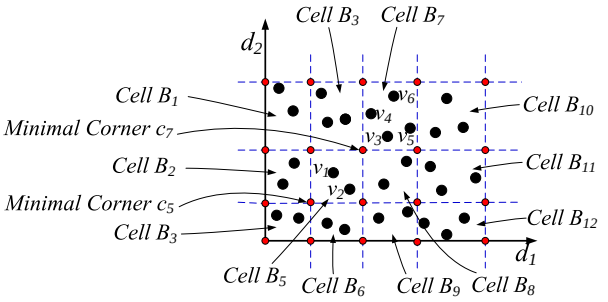


Fig. 4. A partitioning of the data space.

Example 1: The 2-dimensional space in Fig. 4 is partitioned into 12 grid cells $B_1, \ldots, B_{12}$. For instance, the grid cell B_7 contains v_3, v_4, v_5, and v_6. Its minimal corner is c_7.

With minimal corners, we can compute the skyline entities cell by cell, instead of entity by entity.

Definition 3.3: (**Strict Dominance**). Given two corners c_1 and c_2 and a specified numeric attribute set D, we say that c_1 strictly dominates c_2, denoted as $c_1 \prec c_2$, if on each numeric attribute $d_i \in D$, $c_1.d_i < c_2.d_i$ holds.

The strict dominance imposes more stringent restrictions upon two objects. Clearly, if $c_1 \prec c_2$, it holds that $c_1 \prec c_2$, which indicates that strict dominance is a special case of dominating relationship. Moreover, we can derive the following theorem.

Theorem 3.1: Given two minimal corners, c_1 of the grid cell B_1 and c_2 of the grid cell B_2, if $c_1 \prec c_2$, then all the entities in the grid cell B_1 dominate that in B_2.

Proof: For details, please refer to Part A in supplementary materials. □

Example 2: As shown in Fig. 4, since c_5 strictly dominates c_7, i.e., $c_5 \prec c_7$, the entities (v_1 and v_2) in the grid cell B_5 dominate that (v_3, v_4, v_5, and v_6) in B_7.

3.2 Space Partitioning Based Feature Encoding

With the partitioning of the data space, we present a hybrid feature encoding in this subsection.

3.2.1 Structural Feature Encoding

Structural encoding for entities. Provided that u (in q) can match a vertex v (in G), each u's adjacent edge and neighborhood vertex should match some v's adjacent edge and neighborhood vertex, respectively. Thus, if the entities in q and G are encoded in the same method, we can check the match according to their encodings.

Local Structural Encoding. We hash the local structure of an entity v to a bitstring, denoted by $lbStr(v)$, which is similar to, but different from, the previous work [5]. The differences are listed as follows.

- We integrate the adjacent edge and its corresponding neighbor vertex together (denoted by *1-hop* path), instead of considering them separately.
- We add more structural information, i.e., connecting edges (Def. 3.4), to improve the pruning ability.

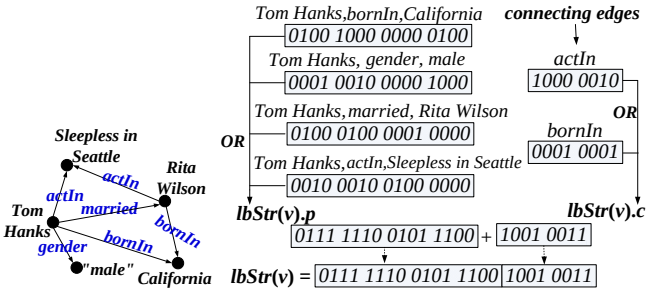
Definition 3.4: **(Connecting Edge).** Given a vertex v and its neighbors $Nei(v)$, v's connecting edges are the edges $e = (v_i, v_j)$ between two neighbor vertices $v_i \in Nei(v)$ and $v_j \in Nei(v)$.

The bitstring of v's local structure $lbStr(v)$ contains two parts: $lbStr(v).p$ and $lbStr(v).c$, where the first part $lbStr(v).p$ denotes the encoding for *1-hop* path labels (v's adjacent edge label combining the corresponding neighbor vertex label), and the second part $lbStr(v).c$ denotes the encoding for connecting edge labels.

Bitstring Generation. Given a neighbor vertex v' of v and the corresponding edge e between v and v', we combine $e.Label$ and $v'.Label$ together to get the label ($p.Label$) of v's *1-hop* path. We generate the bitstring for $p.Label$, i.e., $lbStr(v).p$ ($|lbStr(v).p| = M_1$). We utilize m different hash functions to set m out of M_1 bits in $lbStr(v).p$ to be '1'. All the other bits are set to be '0'. Similarly, we can obtain the other part $lbStr(v).c$.

Example 3: Fig. 5(a) shows the local structure of entity v (Tom Hanks). It has 4 adjacent edges and 2 connecting edges. As shown in Fig. 5(b), $lbStr(v)$ consists of $lbStr(v).p$ and $lbStr(v).c$, which are the unions of the bitstrings for v' 1-hop paths and connecting edges, respectively.

Using the same hash functions we can obtain the local bitstring for each vertex u in q, denoted by $lbStr(u)$. If $lbStr(u)\&lbStr(v) \neq lbStr(u)$, where '&' is the bitwise AND operator, then we can determine that u cannot match v. Therefore, the vertex v can be safely filtered out.



(a) local structure (b) local structural encoding

Fig. 5. The local structure of v and its encoding

Global Structural Encoding. Given a numeric entity v in graph G, we collect the set of numeric entities $NS_h(v)$, such that $dist(v, v') \leq h$ for each $v' \in NS_h(v)$, where $dist(v, v')$ is the shortest path distance between v and v'.

We first generate the bitstring for $v_i.Id$ ($v_i \in NS_h(v)$), denoted as $bStr(v_i)$ ($|bStr(v_i)| = M_2$). Then, we utilize m different hash functions to set m out of M_2 bits in $bStr(v_i)$ to be '1'. All other bits are set to be '0'. Then $gbStr(v)$ is generated by performing bitwise OR operation over the bitstrings of $v_i \in NS_h(v)$, i.e., $gbStr(v) = bStr(v_1)| \ldots |bStr(v_m)$.

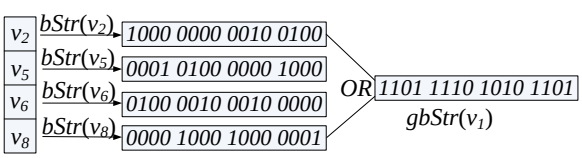


Fig. 6. The global numeric encoding of entity v_1

Example 4: Provided that $NS_3(v_1) = \{v_2, v_5, v_6, v_8\}$, as shown in Fig. 6, we generate the bitstring for each entity in $NS_3(v_1)$, and then perform the OR bitwise operation over these bitstrings to obtain $gbStr(v_1)$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

Structural encoding for grid cells. The intuition is as follows. By summarizing the encodings for the entities in a cell B, denoted as $lbStr(B)$, we can check $lbStr(B)$ before accessing the entities in B. If a query entity does not match $lbStr(B)$, we can safely prune all entities in the grid cell B.

The structural encoding for a grid cell B, i.e., $lbStr(B)$, is formed by performing bitwise OR operation over the local structural bitstrings of the entities in B. Formally, $lbStr(B) = lbStr(v_1)| \ldots |lbStr(v_m)$, where $v_i \in B$ $(1 \leq i \leq m)$. Thus, if $lbStr(u)$ & $lbStr(B) \neq lbStr(u)$, any numeric entity in the grid cell B does not match u, where u is a numeric entity in the query graph q.

3.2.2 Numeric Feature Encoding

The previous method Bitmap [12] can progressively determine whether a point is in the skyline. However, as discussed earlier, it is costly to maintain the Bitmap in terms of storage cost. The numeric encoding $nbStr(B)$ in this paper is distinct from Bitmap [12]:

- We only encode the grid cells rather than the entities. Hence, the size of $nbStr(B)$ is smaller than Bitmap, i.e., $K \cdot |D| < n \cdot |D|$, where K, n, and $|D|$ are the number of cells, numeric entities and attributes, respectively.
- $nbStr(B)$ is online generated if required.

We maintain a bitstring for each cell B, i.e., $nbStr(B)$, which consists of $|D|$ parts: $nbStr(B).d_1, \ldots, nbStr(B).d_{|D|}$.

The size of each part $nbStr(B).d_i$ is K, where K is the number of grid cells. If B_j's value on dimension d_i is better than that of B on dimension d_i, i.e., $B_j.d_i < B.d_i$, the jth bit of $nbStr(B).d_i$ is set to 1, otherwise it is set to 0.

grid Id:	1	2	3	4	5	6	7	8	9	10	11	12
$nbStr(B_5).d_1$	1	1	1	0	0	0	0	0	0	0	0	0
$nbStr(B_5).d_2$	0	0	1	0	0	1	0	0	1	0	0	1

Fig. 7. The numeric encoding of B_5

Example 5: Consider the space partitioning in Fig. 4. The numeric encoding of the grid cell B_5 is shown in Fig. 7.

Obtaining the numeric encoding for B, we can determine whether B is in the skyline on dimensions $d_1, \ldots, d_m$ $(m \leq |D|)$. Let $X = nbStr(B).d_1$ & ... & $nbStr(B).d_m$, where '&' represents the bitwise AND operation. If the result of the operation X is non-zero, we can conclude that there must be a certain cell strictly dominating B.

Similarly, we can also define the numeric encoding for each numeric entity v, denoted by $nbStr(v)$.

4 Subgraph Skyline Computation

In this section, we present an index for grid cells first, and then give the process of computing subgraph skylines.

4.1 Grid Index - Skylayer

Since we partition the data space into grid cells and use cells to represent the entities, the computation of skyline entities is conducted over these cells. We propose skylayer to facilitate the subgraph skyline computation.

Definition 4.1: **(Skylayer)**. Given a set of minimal corners, we organize the corresponding cells in the several layers such that every minimal corner c_i does not strictly dominate any other minimal corner c_j in the same layer.

Example 6: Fig. 8 shows a skylayer example of the dataset in Fig. 4. There are 4 layers: $L_1 \sim L_4$, where L_i maintains the cells that do not strictly dominate each other.

Given the set of minimal corners, C, its skylayer is easy to build by recursively employing any existing skyline algorithms [13], [14], [15]. Specifically, we compute the skyline cells over C to obtain the first layer L_1. Then we remove these cells (in L_1) from C to obtain a new set C', i.e., $C' = C - L_1$. Recursively, we can compute the new skyline cells over C' to get L_i $(i > 1)$ until $C' = \emptyset$.

Theorem 4.1: Any entity v in the layer L_j does not dominate any entity v' in the layer L_i, where $i < j$.

Proof: For details, please refer to Part B in supplementary materials. □

Theorem 4.1 guarantees that accessing the skylayers one by one will not miss any skyline entities.

Obtaining a skyline cell B, we need to compute the skyline entities in B. Here, we employ the bitmap technique [12] to determine whether entity v $(v \in B)$ is in the skyline.

Different from the work in [12], we generate the bitmaps online instead of maintaining all the bitmaps at expensive storage cost. Moreover, it is probable that not all the entities need to be examined, that is, it may only involve a subset of entities. Hence, it is unnecessary to generate bitmaps for all entities. In the offline phase, entities are sorted on each dimension, based on which the bitmap generation is very simple (similar to the generation for cells in Section 3.2.2).

4.2 Subgraph Skyline Computation

The computing process contains three steps: candidate generation (Alg. 2), cell-level pruning (Alg. 3), and entity-level pruning (Alg. 4).

Candidate Generation (Alg. 2). Given a query graph q which contains t numeric entities $u_1, \ldots, u_t$, we generate the local structural encoding $lbStr(u_i)$ for each numeric entity u_i. Then we check whether u_i can match each cell B in the skylayer L. If $lbStr(u)\&lbStr(B) \neq lbStr(u)$, we can conclude that any entity in B can not match u (lines 4-7 in Alg. 2). We also generate the candidate entities in B, denoted by $B.CE(u_i)$, for each numeric entity u_i (lines 8-12 in Alg. 2). Then we compute the subgraph skylines. Based on the discussion in Section 4.1, we explore the cells $\{B_1, \ldots, B_t\}$ layer by layer so that we can find the answers early and reduce the search space.

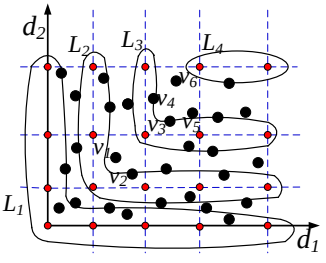


Fig. 8. An example of skylayer.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

Algorithm 2 Subgraph Skyline Computation

Input: A knowledge graph G, the hybrid feature encoding, and a query graph q with t numeric entities $u_1, \ldots, u_t$;
Output: The subgraph skyline in G.

```
1: for each numeric entity u_i ∈ q do
2:    canB(u_i) ← ∅
3:    generate the local structural encoding lbStr(u_i)
4:    for each skylayer L do
5:       for each grid cell B in L do
6:          if lbStr(u_i)&lbStr(B) = lbStr(u_i) then
7:             canB(u_i) ← canB(u_i) ∪ B
8:             B.CE(u_i) ← ∅
9:             for each entity v ∈ B_i do
10:               if lbStr(u_i)&lbStr(v) = lbStr(u) then
11:                  B.CE(u_i) ← B.CE(u_i) ∪ v
12: for each {B_1, ..., B_t} ∈ canB(u_1) ⋈ ... ⋈ canB(u_t) do
13:    GridSkyline(B_1, ..., B_t)
```

Algorithm 3 GridSkyline($B_1, \ldots, B_t$)

Input: The grid cells $B_1, \ldots, B_t$;
Output: The subgraph skyline containing the numeric entities in $B_1 \bowtie \ldots \bowtie B_t$.

```
1: isSL ← 0
2: for 1 ≤ i ≤ t do
3:    X_i ← nbStr(B_i).d_1& ... &nbStr(B_i).d_m
4:    if X_i = 0 then
5:       isSL ← 1
6:       go to line 7
7: if isSL = 1 then
8:    for {v_1, ..., v_t} ∈ B_1.CE(u_1) ⋈ ... ⋈ B_t.CE(u_t) do
9:       isEmpty ← EntitySkyline(v_1, ..., v_t)
10:      if isEmpty = false then
11:         add {B_1, ..., B_t} into VSB
12: else
13:   if {B'_1, ..., B'_t} has not been checked then
14:      GridSkyline(B'_1, ..., B'_t)
15:   if {B_1, ..., B_t} is not dominated by VSB then
16:      for {v_1, ..., v_t} ∈ B_1.CE(u_1) ⋈ ... ⋈ B_t.CE(u_t) do
17:         isEmpty ← EntitySkyline(v_1, ..., v_t)
18:         if isEmpty = false then
19:            add {B_1, ..., B_t} into VSB
```

Grid-level Pruning (Alg. 3). After obtaining the candidate cells, we first compute the skyline cells. The pruning principle is that if a set of cells do not belong to skylines, the entities in the corresponding cells cannot be in the skyline. Before the computation, we introduce valid skyline.

Definition 4.2: **(Valid Skyline.)** Entity v (or cell B) is in valid skyline *iff* 1) v (or B) satisfies the structure constraint specified in the query q, and 2) all the entities dominating v (or B), if any, do not satisfy the structure constraint.

For instance, assume that B_1 dominates B_2, whereas B_1 does not satisfy the structure constraint, and there exist no other cells that dominate B_2. Thus, B_2 is in the valid skyline, and B_1 is an invalid skyline cell.

Alg. 3 gives the details to compute the valid cell skylines. If the set of cells $\{B_1, \ldots, B_t\}$ is not dominated by any other cells, we explore the entities in the corresponding cells. The set of cells $\{B_1, \ldots, B_t\}$ is added into the valid skylines VSB on condition that we find some subgraph skylines utilizing the entities in $\{B_1, \ldots, B_t\}$ (lines 7-11).

If the set of cells $\{B_1, \ldots, B_t\}$ is dominated by another set of cells $\{B'_1, \ldots, B'_t\}$, we should determine whether $\{B_1, \ldots, B_t\}$ is in the valid skyline. If $\{B'_1, \ldots, B'_t\}$ has not been checked, the procedure GridSkyline($B'_1, \ldots, B'_t$) is invoked (lines 13-14). If the set $\{B_1, \ldots, B_t\}$ is not dominated by the cells in VSB, we need to explore the entities in $B_1.CE(u_1) \bowtie \ldots \bowtie B_t.CE(u_t)$ (lines 15-19).

Algorithm 4 EntitySkyline($v_1, \ldots, v_t$)

Input: The numeric vertices $v_1, \ldots, v_t$;
Output: The subgraph skyline containing $v_1, \ldots, v_t$.

```
1: isSL ← 0
2: for 1 ≤ i ≤ t do
3:    X_i ← nbStr(v_i).d_1& ... &nbStr(v_i).d_m
4:    if X_i = 0 then
5:       isSL ← 1
6:       go to line 7
7: if isSL = 1 then
8:    perform the shortest-path-distance pruning
9:    if a graph g containing {v_1, ..., v_t} is isomorphic to
      q then
10:      report g as a result
11:      add {v_1, ..., v_t} into VSV
12: else
13:   if {v'_1, ..., v'_t} has not been checked then
14:      EntitySkyline(v'_1, ..., v'_t)
15:   if {v_1, ..., v_t} is not dominated by VSV then
16:      perform the shortest-path-distance pruning
17:      if a graph g containing {v_1, ..., v_t} is isomorphic
         to q then
18:         report g as a result
19:         add {v_1, ..., v_t} into VSV
```

Entity-level Pruning (Alg. 4). For a set of entities $\{v'_1, \ldots, v'_t\}$, we first check whether it is dominated by any other entities (lines 2-6). If it is not dominated by any other entities, we perform the structure verification. The subgraph g is reported as an answer if it is graph isomorphic to q and contains the entities $\{v_1, \ldots, v_t\}$ (lines 7-11).

Analogous to that in the grid-level computation, if the set of entities $\{v_1, \ldots, v_t\}$ is dominated by another set of entities $\{v'_1, \ldots, v'_t\}$, we should determine whether $\{v_1, \ldots, v_t\}$ is in the valid skyline. If $\{v'_1, \ldots, v'_t\}$ has not been checked, the procedure EntitySkyline($v'_1, \ldots, v'_t$) is invoked (lines 13-14). If the set $\{v_1, \ldots, v_t\}$ is not dominated by the entities in VSV, we need to verify the structure (lines 15-19). The state-of-the-art algorithms such as Ullmann [16] and VF2 [9] can be employed to perform this verification.

To improve the efficiency, we propose a shortest-path-distance pruning technique before verifying the structure. Provided that the shortest path distance between u_1 and u_2, $dist(u_1, u_2)$, is no larger than h, where h is a predefined threshold based on which the global structural

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

encoding is generated. Entity v_2 matches u_2 only if $v_2 \in NS_h(v_1)$. It is easy to determine whether v_2 belongs to $NS_h(v_1)$ using the global structure encoding $gbStr(v_1)$. If $gbStr(v_1)\&bStr(v_2) \neq bStr(v_2)$, we can conclude that the candidate pair (v_1, v_2) does not match (u_1, u_2), where $bStr(v_1)$ is the bitstring of $v_2.Id$.

5 Optimization on Space Partitioning

Since different partitionings result in different pruning effects, we discuss how to obtain a good partitioning below.

5.1 What is a Good Partitioning?

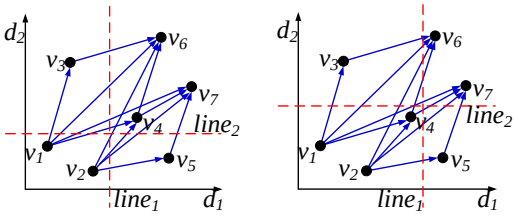
Since it is required to check whether all the entities in candidate cells are valid skyline entities, the less false positives are generated, the better the partitioning will be.

Observation 1: A good partitioning should generate less false positives.

Definition 5.1: **(Dominating Edge)**. If entity v_1 dominates entity v_2, a directed edge starting from v_1 to v_2 is added. This directed edge is named as dominating edge.

Using the dominating edges, we can build a dominating graph.

Definition 5.2: **(Dominating Graph)**. Each vertex in the dominating graph is an entity. If vertex v_1 dominates vertex v_2, there is a directed edge from v_1 to v_2, i.e., the dominating edge between v_1 and v_2.



(a) Partitioning 1 (b) Partitioning 2

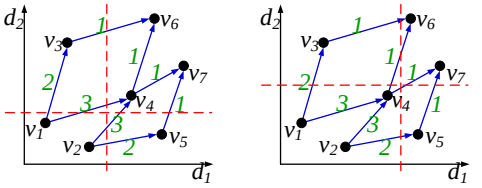
Fig. 9. Two partitions for a dominating graph.

Fig. 9 shows a dominating graph and there are two different partitionings for the data space. However, it is hard to distinguish the effect of these two partitionings since both of them destroy 6 common dominating relations. Actually, the partitioning in Fig. 9(a) is better than that in Fig. 9(b), because the cell consisting of v_1, v_2, and v_4 only prunes two entities (i.e., v_6 and v_7) in Fig. 9(b). In contrast, the cell consisting of v_1 and v_2 in Fig. 9(a) can prune three entities, i.e., v_4, v_6, and v_7. Hence, we propose the weighted dominating graph. Let $M(v)$ denote the set of dominated entities by v, $T(v)$ denote the set of entities dominating v.

Definition 5.3: **(Weighted Dominating Graph**, denoted by $\mathbb{T}_D$). Consider each two entities v_i and v_j in $T(v)$ in a dominating graph. If v_i dominates v_j, the dominating edge between v_i and v is removed. Each left edge $e(v_i, v_j)$ is assigned with a weight $(1+|M(v_j)|)$.

Example 7: Consider the data space in Fig. 9(a). Its weighted dominating graph is shown in Fig. 10(a).

Given a partitioning P, we can sum up the weights of the common destroyed edges, denoted by $W(P) = \sum_{e \in U} w(e)$, where U is the set of common destroyed edges and $w(e)$ is the edge weight.



(a) Partitioning P_1 (b) Partitioning P_2

Fig. 10. Two partitionings for $\mathbb{T}_D$.

Using the same partitioning lines in Fig. 9, we partition the weighted dominating edges in Fig. 10. It is clear that partitioning P_1 in Fig. 10(a) is better than partitioning P_2 in Fig. 10(b) because $W(P_1) = 6 \geq W(P_2) = 2$. Therefore, we obtain the following observation.

Observation 2: An effective partitioning P should generate larger $W(P)$.

It has two advantages to utilize weighted dominating graphs: (1) the pruning power is improved as discussed above; (2) the computation cost is reduced benefiting from the fewer dominating edges.

In real applications, we can determine the number of cells based on storage cost. For simplicity, we assume that the number of partitioning lines are given in the discussion.

Definition 5.4: (**Maximum Weighted Common Partitioning**) (denoted as **MWCP**). Given the number of partitioning lines on each dimension and a space partitioning P, if there exist no other partitioning P' such that $W(P) < W(P')$, P is the maximum weighted common partitioning.

Given a data space, it is better to find the maximum weighted common partitioning to improve the pruning power. However, it has been proven to be NP-hard.

Theorem 5.1: Given a set of data, computing the maximum weighted common partitioning is NP-hard.

Proof: For details, please refer to Part C in supplementary materials. □

5.2 Greedy Partitioning

The projections of all the dominating edges on dimension d_i form a universe set E_i. A partitioning line between $v_j.d_i$ and $v_{j+1}.d_i$ destroys a subset of E_i.

Example 8: There are 6 optional partitioning lines in Fig. 11. The partitioning line between $v_3.d_1$ and $v_2.d_1$ destroys 2 edges, i.e., $e(v_1, v_4)$ and $e(v_3, v_6)$. Hence, the corresponding subset is $\{[e(v_1, v_4), 3], [e(v_3, v_6), 1]\}$. Similarly, we can also obtain the other 5 subsets.

The intuition is: Since it requires destroying more common edges (edges that are destroyed on all dimensions) with larger weights, we should intersect the selection on dimension d with the current selected set U. More details are presented in Alg. 5, which consists of three steps.

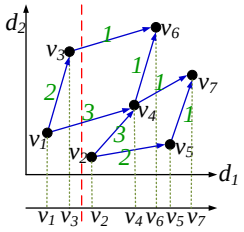


Fig. 11. Projections on dimension d_1.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

- We obtain the family of dominating edge sets $F_i = \{E^i_1, \dots E^i_{n-1}\}$ on each dimension d_i (lines 1-5). Assume that $v_1.d_i < v_2.d_i < \dots < v_n.d_i$. Each set E^i_j (corresponding to the interval $[v_j.d_i, v_{j+1}.d_i]$) consists of the edges in $e(v_j, *) \cup E^i_{j-1} \setminus e(*, v_j)$, where $e(v_j, *)$ is the set of edges starting from vertex v_j and $e(*, v_j)$ is the set of edges ending at v_j. Initially, $E^i_1 = e(v_j, *)$.
- The set $E(\in F_i)$ with the largest weight is selected. Then we remove all elements $e \in E$ from the remaining sets, and select the largest-weight set from the updated sets in next loop. The selection process terminates until k_1 sets have been selected. The union of these selected sets is denoted as U (lines 6-12).
- Considering the next dimension d_j, we select the set $E \in F_j$ such that the intersection of E and U (i.e., $|E \cap U|$) has the largest weight (line 16). Then we remove all the elements $e \in E$ from the remaining sets (lines 17-18). The selection process terminates until k_i sets have been selected. U is updated with the intersection of R and U, i.e., $U \cap R$. The algorithm stops when all dimensions have been considered (lines 13-21).

Algorithm 5 Greedy Partitioning

Input: The numeric entities in dimensions $D = \{d_1, \dots d_{|D|}\}$, the weighted dominating graph $\mathbb{T}_D$, and the number of partitioning lines k_i on each dimension.
Output: A partitioning of the data space.

```
1: for Each dimension d_i ∈ D do
2:   for 1 ≤ j ≤ n − 1 do
3:     put the edges e(v_j, *) ∪ E^i_{j−1} into E^i_j
4:     remove the edge e(*, v_j) from E^i_j
5:     put E^i_j into F_i
6: U ← ∅ , s ← 0
7: while s < k_1 do
8:   select the largest set E^1 ∈ F_1
9:   for each E^1_j ∈ F_1 do
10:    remove all the elements e ∈ E^1 from E^1_j
11:  U ← U ∪ E^1 , s ← s + 1
12:  remove E^1 from F_1
13: for each dimension d_i ∈ D ∧ i ≠ 1 do
14:   s ← 0, R ← ∅
15:   while s < k_i do
16:     select the set E^i ∈ F_i with the largest E^i ∩ U
17:     for each E^i_j ∈ F_i do
18:       remove all the elements e ∈ E^i from E^i_j
19:     R ← R ∪ E^i , s ← s + 1
20:     remove E^i from F_i
21:   U ← U ∩ R
```

Time complexity. In the first step (lines 1-5), we generate the dominating sets F_i for each dimension. Since each set E^i_{j-1} contains n^2 edges at most, it requires $O(n^2)$ to remove the edges $e(*, v_j)$ from E^i_j. Thus, the time complexity of the first step is $O(n^3 \cdot |D|)$. The time complexity of selecting the largest set and updating the set F_i is $O(n^3)$. Thus, the time complexity of selecting k sets on a dimension is $O(n^3 \cdot k)$. Hence, the overall time complexity of Alg. 5 is $O(n^3 \cdot |D| \cdot k)$.

6 High Dimensionality

In a heterogenous knowledge graph, there are enormous numeric attributes associated with entities of different types. Thus, the weighted dominating graph is usually flat, which may result in too many cells in each single layer and diminish the pruning ability definitely. This phenomenon is actually the curse of dimensionality.

Note that entities in a knowledge graph G are naturally distinguished by types. More importantly, it is not required to check the dominating relation between entities of different types. Hence, it is reasonable to build index for entities of each type separately. Hereon, all the following discussions focus on the entities of a certain type.

6.1 Attribute Cluster

We propose a clustering method to solve the curse of dimensionality. The main idea is: Given the set of attributes $D = \{d_1, d_2, \dots, d_{|D|}\}$, we divide the attributes into several subsets $\mathbb{C} = \{D_1, D_2, \dots, D_k\}$, where $D_i \subseteq D$ $(1 \le i \le k)$, and the subsets are disjoint. Then we construct the corresponding weighted dominating graph over each subset D_i.

We design two metrics to measure the cluster as follows.

Cluster coverage. Considering a query q given online, the numeric attributes involved in q, denoted by $D(q)$, are expected to be covered by a cluster D_i, that is $D(q) \subseteq D_i$, so that the query can be searched in a single cluster D_i.

In a universe of numeric attributes, i.e., the full attribute space D, there are $(2^{|D|} - 1)$ possible attribute subspaces that may be involved in queries. For simplicity, we assume that each attribute subspace is queried with identical probability.

Definition 6.1: (**Cluster Coverage Rate**). Given a clustering result $\mathbb{C} = \{D_1, D_2, \dots, D_k\}$ over the full attribute space D, its cluster coverage rate is the proportion of possible attribute subspaces covered by all the clusters, formally defined as Equation 1.

$$ccr(\mathbb{C}) = \frac{\sum_1^k (2^{|D_i|} - 1)}{2^{|D|} - 1} \tag{1}$$

Consider an extreme case that $k = 1$, i.e., there is only one cluster D. The cluster coverage rate achieves the maximum value. However, the weighted dominating graph may be flat, which diminishes the pruning ability. Hence, we propose dominating degree to evaluate a cluster.

Dominating degree. Generally, each two numeric entities should have a dominating/non-dominating relationship. With the proportion of dominating edges increasing, the flatness of the weighted dominating graph will decrease.

Definition 6.2: (**Dominating Degree**). Given a clustering result $\mathbb{C} = \{D_1, D_2, \dots, D_k\}$ over the full attribute space D, its dominating degree is formally defined as Equation 2,

$$dd(\mathbb{C}) = \frac{\sum_1^k (2 \cdot S(D_i))}{k \cdot n \cdot (n-1)} \tag{2}$$

where $S(D_i)$ is the summation of all the edge weights in the weighted dominating graph over subspace D_i, n is the number of numeric entities.

Note that the dominating degree achieves the maximum when $k = |D|$, that is, each numeric attribute forms a cluster.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

Combining both cluster coverage rate and dominating degree, we give the following function to evaluate a clustering result $\mathbb{C}$, where α is a constant.

$$cd(\mathbb{C}) = \alpha \cdot ccr(\mathbb{C}) + (1-\alpha) \cdot dd(\mathbb{C}) \quad (3)$$

Given a set of numeric attributes D, it is expected to obtain $\mathbb{C}$ with the maximum $cd(\mathbb{C})$. However, we have proved that it is an NP-hard problem.

Theorem 6.1: Given a set of numeric attributes D, and the number of clusters k, finding $\mathbb{C}$ with the maximum $cd(\mathbb{C})$ is NP hard.

Proof: For details, please refer to Part D in supplementary materials. □

6.2 Automatic Clustering

The high-level idea: Initially, we set $k = |D|$, that is, the entities on each dimension d_i are sorted in the non-decreasing order. Then, we perform a hierarchical clustering.

Since it requires constructing the weighted dominating graph to compute the dominating degree in each step, the clustering process is not efficient enough. Thereupon, we propose an LCS (longest common subsequence)-based method to avoid building the weighted dominating graph.

Definition 6.3: (Longest Common Subsequence). Given several sequences $s_1, s_2, \cdots, s_t$, their longest common subsequence, denoted by LCS, is the longest subsequence $a_{i_1}, a_{i_2}, \cdots, a_{i_m}$ common to all sequences.

Note that elements in the longest common subsequence are not required to occupy consecutive positions within the original sequences.

Assume $V_{d_i} = \{v_{i_1}, v_{i_2}, \ldots, v_{i_n}\}$ is the set of sorted numeric entities over dimensions d_i. If v_1 dominates v_2 over the subspace $\{d_i, d_j\}$, it indicates that v_1 is in front of v_2 in both V_{d_i} and V_{d_j}. Thus we have the following theorem.

Theorem 6.2: The height of the weighted dominating graph $\mathbb{T}_{D_1}$ over subspace $D_1 = \{d_1, \ldots, d_{|D_1|}\}$ corresponds to the longest common subsequence of $V_{d_1}, \ldots, V_{d_{|D_1|}}$.

Proof: For details, please refer to Part E in supplementary materials. □

Example 9: Let us consider the entities in Fig. 10. We project these entities on dimensions d_1 and d_2 to obtain two sorted sequences $V_{d_1} = \{v_1, v_3, v_2, v_4, v_6, v_5, v_7\}$ and $V_{d_2} = \{v_2, v_5, v_1, v_4, v_7, v_3, v_6\}$. An LCS of V_{d_1} and V_{d_2} is $\{v_1, v_4, v_7\}$, which indicates the height of the weighted dominating graph over $\{d_1, d_2\}$ is 3.

In order to diminish the flatness, two dimensions, on which the sorted sequences have larger $|LCS|$, should be clustered together.

For ease of presentation, let $LCS_v(D_1)$ denote the longest common subsequence starting from the vertex v. Actually, $|LCS_v(D_1)|$ corresponds to the height of the subgraph rooted at vertex v in $\mathbb{T}_{D_1}$,

We propose the LCS-based score to determine the dominating degree of a cluster.

Definition 6.4: The LCS-based dominating degree of the subspace D_i is defined as follows:

$$dd_{LCS}(D_i) = \frac{\sum_v 2 \cdot |LCS_v(D_i)|}{n \cdot (n-1)} \quad (4)$$

Therefore, we can use the following equation to evaluate a clustering result $\mathbb{C}$ efficiently.

$$\widetilde{cd}(\mathbb{C}) = \alpha \cdot ccr(\mathbb{C}) + (1-\alpha) \cdot \frac{\sum_{D_i \in \mathbb{C}} dd_{LCS}(D_i)}{k} \quad (5)$$

Before presenting the clustering details, we discuss how to compute $|LCS_v(D_i)|$. As shown in Algorithm 6, we first initialize each $|LCS_{v_j}(D_i)|$ as 1. Then we compute $h_t(v_j)$ for each V_{d_t} $(t > 1)$, where $h_t(v_j)$ represents the set of vertices whose position number in V_{d_t} are larger than that of v_j. The intersection of sets $h_2(v_j) \ldots \cap h_{|D_i|}(v_j)$ is denoted by $H(v_j)$. Let max denote the maximum $|LCS_v(D_i)|$ for all vertices $v \in H(v_j)$. Adding max to $|LCS_{v_j}(D_i)|$ will lead to $|LCS_{v_j}(D_i)|$. Repeating the procedure above, we can obtain $|LCS_v(D_i)|$ for all vertices $v \in V_d$.

Algorithm 6 $LCS_v(D_i)$

Input: $V_{d_1}, V_{d_2}, \cdots, V_{d_{|D_i|}}$;
Output: $|LCS_{v_1}(D_i)|, |LCS_{v_2}(D_i)|, \cdots, |LCS_{v_n}(D_i)|$.

```
1: for Each vertex v in V_{d_1} do
2:     |LCS_v(D_i)| ← 1
3: for Each v_j ∈ V_{d_1} (j from n − 1 to 1) do
4:     for Each t from 2 to |D_i| do
5:         compute h_t(v_j)
6:     H(v_j) ← h_2(v_j) ∩ ... ∩ h_{|D_i|}(v_j)
7:     max ← max_{v∈H(v_j)} |LCS_v(D_i)|
8:     LCS_{v_j}(D_i) ← |LCS_{v_j}(D_i)| + max
9: return |LCS_{v_1}(D_i)|, |LCS_{v_2}(D_i)|, ..., |LCS_{v_n}(D_i)|
```

Using Equation 5 we devise an efficient clustering algorithm as presented in Algorithm 7.

Initially, $\mathbb{C} = \{\{d_1\}, \ldots, \{d_i\}, \ldots, \{d_j\}, \ldots, \{d_n\}\}$, i.e., each dimension is a cluster. First, we compute $dd_{LCS}(\{D_i, D_j\})$ for each pair of subspaces D_i and D_j (lines 2-3). The subspaces with the largest $dd_{LCS}(\{D_i, D_j\})$ are clustered together. Then we update $dd_{LCS}(\{\cdot, \cdot\})$ for each pair of subspaces. Note that before continuing the clustering process, we compute the $\Delta = \widetilde{cd}(\mathbb{C}') - \widetilde{cd}(\mathbb{C})$, where $\mathbb{C}'$ is the newly generated cluster (lines 9-10). The process proceeds recursively until $\Delta \leq 0$.

Algorithm 7 Automatic Clustering

Input: $\mathbb{C} = \{\{d_1\}, \ldots, \{d_i\}, \ldots, \{d_j\}, \ldots, \{d_{|D|}\}\}$;
Output: $\mathbb{C} = \{D_1, D_2, \ldots, D_k\}$.

```
1: Δ = 1
2: for Each two subspaces D_i and D_j in ℂ do
3:     compute dd_LCS({D_i, D_j})
4: while Δ > 0 do
5:     ℂ ⟵ ℂ′
6:     subspace pair (D_i, D_j) ⟵ arg max dd_LCS({D_i, D_j})
7:     combine D_i and D_j to form a subspace D_ij = {D_i, D_j}
8:     let ℂ′ be the new clustering result
9:     compute cd̃(ℂ) and cd̃(ℂ′)
10:    Δ = cd̃(ℂ′) − cd̃(ℂ)
11:    compute each dd_LCS({D_ij, ·})
12: return ℂ
```

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

Time Complexity. The time complexity of the initial phase (lines 2-3) is $O(|D|^2 \cdot n^2)$, where n is the number of entities. Since it requires $O(n^2|D| \log n)$ to compute $dd_{LCS}(\{D_{ij}, \cdot\})$, the time complexity of each iteration (lines 5-11) is $O(n^2|D|^2 \log n)$ in the worst case. Hence, the time complexity of automatic clustering (Algorithm 7) is $O(n^2|D|^3 \log n)$.

6.3 Query Processing With Clusters

In order to solve the curse of dimensionality, we organize the numeric attributes into several subspaces (i.e., clusters) for entities of each type and build a skylayer for each subspace. If the numeric attributes of an entity in the query q are covered by a subspace, the methods presented in Section 4 can be utilized directly. Otherwise, it requires several subspaces to cover the entity. Here, we design an efficient strategy to determine the skyline candidates.

Theorem 6.3: If an entity v is in the skyline over the subspace D_i, it must be in the skyline over the subspace D_j, where $D_i \subseteq D_j$.

Proof: For details, please refer to Part F in supplementary materials. □

The procedure starts from the first layer of a subspace D_i that covers one or more dimensions of $u \in q$. If the candidate entity v is in the skyline over D_i, we check the structure constraint directly. Otherwise, we need to check whether v is in skyline on the dimensions involved in u. All the candidates dominated by v are denoted as $M_v(D_i)$. The candidates in the intersection of all $M_v(D_i)$ can be pruned.

7 Experimental Study

In this section, we evaluate our proposed method through extensive experiments.

7.1 Experiment Setup

We use two real datasets in our experiments.

Dataset 1. We use the dataset Freebase[1] which integrates *NBA*[2] and *IMDB*[3] to evaluate our method. It contains 12,130,534 vertices, 232,671,328 edges, 7,634,315 numeric entities, and 35 numeric attributes.

Dataset 2. DBpedia is a knowledge base derived from Wikipedia[4] to support sophisticated queries. It has 870 numeric attributes, such as "populationDensity", "height", "numberOfPages", "areaLand", and ,"runtime". This dataset contains 3,220,134 vertices, 40,504,436 edges, and 1,422,102 numeric entities.

Regarding the queries, we generate some query graphs to study the effectiveness of our method. More examples will be presented in Section 7.2. In order to study the efficiency, we randomly extract some subgraphs containing numeric entities, and vary the size of these query graphs.

All the experiments were conducted on a PC with 2.9GHz CPU and 16GB main memory running Linux operating system. For comparison, we implement the simple methods presented in Section 2.2, denoted as *"NaiveIso"* and *"NaiveJoin"*, respectively. Our newly proposed cluster-based method in this paper is denoted as "*parCode*$^+$". We also compare it with the previous method "*parCode*" [17], which does not deal with the curse of dimensionality. All programs were implemented in C++.

1. http://www.freebase.com/
2. http://databasebasketball.com.
3. http://www.imdb.com/interfaces.
4. http://en.wikipedia.org/wiki/Main_Page.

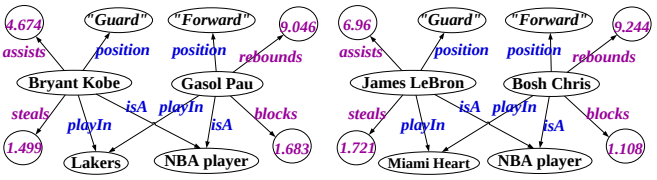


Fig. 12. Golden basketball partner finding.

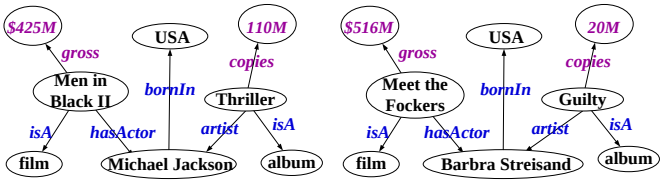


Fig. 13. Excellent versatile artist finding.

7.2 Effectiveness Evaluation

In order to verify the effectiveness of our method, we first give two case-study examples followed by the user study.

Golden Basketball Partner Finding. As mentioned in Section 1.1, assume that we want to find one guard and one forward who play in the same team and have excellent techniques. The query graph is shown in Fig. 2. Fig. 12 presents a subset of the results. As expected, we find several great partners, such as Gasol Pau and Bryant Kobe.

Excellent Versatile Artist Finding. We want to seek American outstanding artist who is both a singer with a large number of his/her album copies and an actor/actress with high gross of his/her films (as shown in Fig. 3). Fig. 13 gives a fraction of the results. For example, Michael Jackson is in the skyline.

Before presenting the details of the user study, we report the average number of subgraph matches (i.e., the subgraphs matching the query q without considering the numeric constraint, denoted by #*subgraphMatch*) and subgraph skyline answers (denoted by #*subgraphSkyline*) by varying the number of numeric entities. As shown in Table 2, both #*subgraphMatch* and #*subgraphSkyline* increase with the growth of the number of entities. Note that #*subgraphSkyline* is much smaller than #*subgraphMatch*. If the number of subgraph skyline answers is still too large, we can define the *top-k* skyline answers as returning k subgraphs with the largest dominating score, where the dominating score of a subgraph g can be defined as the number of subgraphs that are dominated by g.

User study. Since answers in this paper have no order, we utilize the metric Normalized Cumulative Gain ($NCG@k$) to evaluate the returned results. $NCG@k = \frac{1}{|Q|} \sum_{q \in Q} Z_k \sum_{i=1}^{k} r_i$, where Q is the set of queries, r_i is the score of the result at position i, and Z_k is a normalization term to let the perfect match have score 1. As suggested in [18], we set r_i to 3

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering

TABLE 2
The number of results

Dataset	*Freebase*			*DBpedia*		
#numeric entities	1	2	3	1	2	3
#subgraphMatch	712	1,307	2,273	216	658	1,052
#subgraphSkyline	13	37	92	9	32	61

TABLE 3
Manual evaluation ($NCG@k$)

Dataset	*Freebase*			*DBpedia*		
$NCG@k$	$k=3$	$k=5$	$k=7$	$k=3$	$k=5$	$k=7$
subgraphMatch	0.262	0.333	0.396	0.318	0.354	0.427
subgraphSkyline	0.851	0.866	0.877	0.865	0.882	0.904

TABLE 4
Storage cost (MB)

Datasets	*NaiveIso*	*NaiveJoin*	*parCode*	*parCode*$^+$
Freebase	7,342	458	873	1,634
DBpedia	1,593	76	205	782

for the good match, 1 for the relevant match and 0 for the bad match. To simplify the evaluation, we randomly select k answers from the complete answers. 50 queries are constructed for both Freebase and DBpedia datasets. 10 students help evaluate the answers including subgraph matches and subgraph skylines. The students assign each answer with a label, i.e., good, relevant or bad, according to their knowledge. Table 3 presents the quality of the returned answers. The results confirm the usefulness of subgraph skylines.

7.3 Efficiency Evaluation

In this subsection, we evaluate the performance of our proposed method and compare it with its competitors.

7.3.1 Offline Performance

Evaluate storage cost. Table 4 shows the storage costs of these methods. Since *NaiveIso* computes the bitmap for each numeric entity and maintains these bitmaps, it consumes the most storage. *NaiveJoin* does not utilize complicated structure feature. Hence, its space cost is the lowest. Our newly proposed method *parCode*$^+$ consumes more space than *parCode*, because *parCode*$^+$ adopts cluster-based technique to handle the curse of dimensionality.

Evaluate index building time. Table 5 gives the index building time over Freebase and DBpedia. It is obvious that *NaiveIso* requires the most time, since it encodes all the numeric entities one by one. In contrast, instead of encoding each entity, *parCode* just encodes the minimal corner of each cell. What is more, the time consumed by *parCode* is much less than that consumed by *NaiveIso*. *parCode*$^+$ consumes more time compared with *parCode* due to optimizations on high dimensionality.

7.3.2 Online Performance

In this subsection, we adopt two metrics, i.e., the query response time and pruning power, to evaluate the online performance, where the pruning power is the ratio of

TABLE 5
Index building time (s)

Datasets	*NaiveIso*	*NaiveJoin*	*parCode*	*parCode*$^+$
Freebase	23,452	614	5,011	7,324
DBpedia	6,675	102	918	1,823

candidates that are filtered out, i.e., the number of pruned entities divided by all candidates.

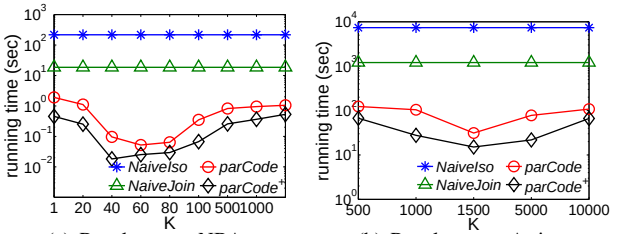
Evaluate the effect of K (the number of cells). We fix the query size (the number of vertices) of q to be 8, and vary the number of cells. Each query may contain one or multiple numeric entities. Both the query response time and pruning power are averaged over all results.

As discussed in Section 6, we build index for each type of vertex separately. Figs. 14(a) and 14(b) investigate the query response time of the two algorithms with respect to NBA and artist analyses over Freebase. It shows that when K is too small or large, the response time increases. Extremely, if $K = 1$ or $K = n$, it is equivalent to the case without any partitioning in actual. According to this experiments, it indicates that K is better to be about $\sqrt{n}$. Since *NaiveIso* and *NaiveJoin* are independent of partitionings, their query response time are horizontal lines.

Fig. 15 gives the results over two types of entities "city" and "person" (in DBpedia), respectively. Similar to that in Fig. 14, *parCode*$^+$ is the most efficient. Note that the gap between *parCode* and *parCode*$^+$ in Fig. 15 is larger than that in Fig. 14. The main reason is that there are more numeric attributes in DBpedia, and *parCode* does not adopt any techniques to deal with the curse of dimensionality.

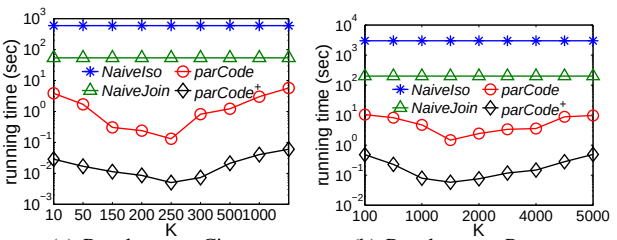
We also study the effect of K on pruning power, i.e., the ratio of entities that are filtered out. As shown in Fig. 16, *parCode*$^+$ has stronger pruning ability than *parCode* especially over the entities of type person, which indicates that our attribute cluster-based method is effective.

Evaluate the effect of N_q. We fix the number of cells, and vary the number of numeric entities, N_q, in q from 1 to 5.



(a) Results w.r.t. NBA (b) Results w.r.t. Artist

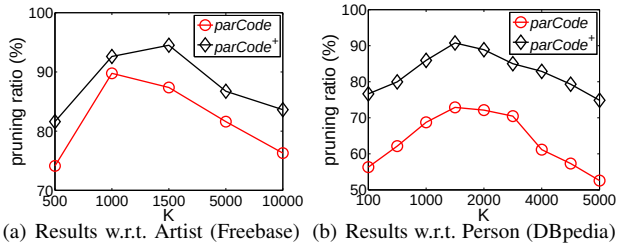
Fig. 14. Query response time vs. K (Freebase).



(a) Results w.r.t. City (b) Results w.r.t. Person

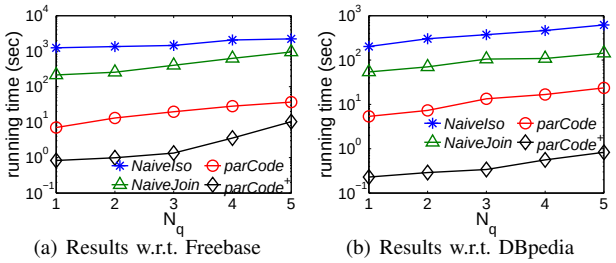
Fig. 15. Query response time vs. K (DBpedia).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2530063, IEEE Transactions on Knowledge and Data Engineering



(a) Results w.r.t. Artist (Freebase) (b) Results w.r.t. Person (DBpedia)

Fig. 16. Pruning power vs. K.



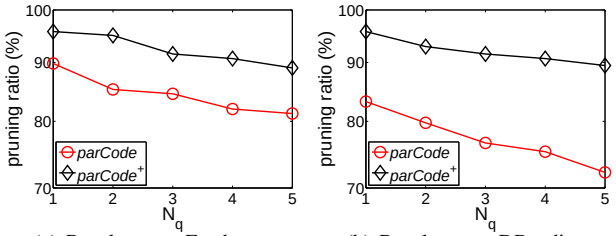
(a) Results w.r.t. Freebase (b) Results w.r.t. DBpedia

Fig. 17. Query response time vs. N_q.

As depicted in Fig. 17, the response time of all methods increase with the growth of N_q. Our newly proposed method *parCode*$^+$ outperforms *NaiveIso* and *NaiveJoin* orders of magnitudes in terms of time efficiency. Furthermore, the performance gap between *parCode*$^+$ and *parCode* becomes larger over the DBpedia dataset as shown in Fig. 17(b).

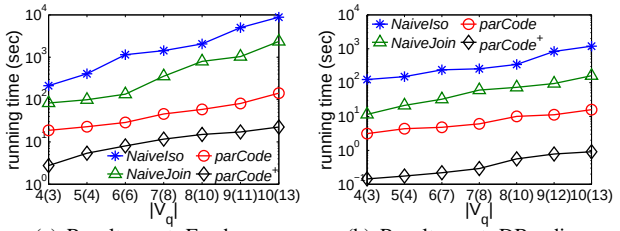
In order to study the pruning power of these two methods *parCode* and *parCode*$^+$, we vary N_q from 1 to 5. As shown in Fig. 18, most of the candidates are pruned without invoking subgraph isomorphism verification, which contributes to the efficiency of our method.

When N_q is larger, the pruning ability of *parCode* over DBpedia degrades greatly. It is because that there are more numeric attributes in DBpedia and the index of *parCode* becomes ineffective. In comparison, *parCode*$^+$ utilizes the weighted dominating graph and attribute cluster based techniques. Hence, its pruning effect is much better. Evaluate the effect of $|V(q)|$. We study effect of the number of vertices in q. Fig. 20 shows that the time consumed of each method increases with the growth of $|V(q)|$ (the edge number is enclosed in parenthesis beside the vertex number). Obviously, if there are more vertices in q, more time will be consumed by the subgraph isomorphism checking. Evaluate the effect of $|E(q)|$. Fixing the number of vertices ($|V(q)| = 7$), we vary the number of edges from 6 to 14. As presented in Fig. 20, although the time efficiency of all methods decreases when there are more edges, *parCode*$^+$ is still the most efficient, which confirms the superiorities



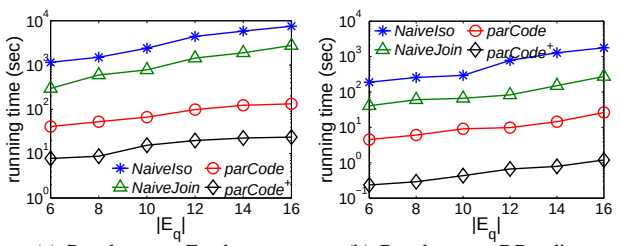
(a) Results w.r.t. Freebase (b) Results w.r.t. DBpedia

Fig. 18. Pruning power vs. N_q.



(a) Results w.r.t. Freebase (b) Results w.r.t. DBpedia

Fig. 19. Query response time vs. $|V(q)|$.



(a) Results w.r.t. Freebase (b) Results w.r.t. DBpedia

Fig. 20. Query response time vs. $|E(q)|$.

of our proposed techniques.

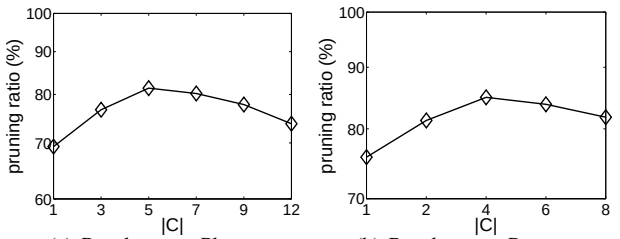
Evaluate the effect of $|\mathbb{C}|$. We vary the effect of the number of clusters $|\mathbb{C}|$ to study its effect on the pruning power. Figure 21 shows the results over places and person. When $|\mathbb{C}| = 1$, i.e., all numeric attributes are clustered in a single cluster, the pruning effect is the worst since dominating degree is the smallest in this case. On the other hand, if each numeric attribute is clustered separately (i.e., $|\mathbb{C}| = 12$ and $|\mathbb{C}| = 8$), the pruning power degrades as well.

8 Related Work

In this section, we briefly review previous works on the skyline computation and subgraph search.

Skyline Computation. Most existing skyline literature focus on multi-dimensional relational data. Their inputs are relational tables. They can be classified into two categories, i.e., *single-relation skyline* and *join-based skyline*.

BNL [13] is a block-nested-loops algorithm to compute the skyline. Its main idea is: each point p is compared with every other point p', and p is reported as a skyline result if there exists no any other point that dominates p. Based on the same principle, *DC* [13] divides the data space into several regions, and produces the final results from points in the regional skylines. Instead of going over the entire dataset, Bitmap [12] represents a data point p using an m-bit vector, which encodes the points having no smaller coordinate than p on each dimension. Then the skyline points can be progressively obtained by performing bitwise



(a) Results w.r.t. Place (b) Results w.r.t. Person

Fig. 21. Pruning power vs. $|\mathbb{C}|$ (DBpedia).

operation over the corresponding bitmaps. However, the storage cost of Bitmap is $O(n^2)$, which is costly especially when there are a large number of points.

Both *NN* [19] and *BBS* [20] compute the skyline exploiting an R-tree. In order to answer subspace skyline queries, *SUBSKY* [14] converts each multi-dimensional point to 1D values, and index these converted values by a single B-tree. Jin et al. [21] pre-computes the "better" and "equal" subspaces for each pair of objects, and group pairs that share the same subspaces into maximal space index. In practice, this method still suffers from expensive storage cost. Similar to the relational aggregation operator "data cube" [22], Pei et al. [23] and Yuan et al. [24] independently propose the sky cube, which consists of skylines in all possible non-empty subspaces. In summary, these algorithms are custom devised for a single relation. Thus, it is oblivious to directly employ them to solve S^2A.

Given two tables R_1 and R_2 with a set of join attributes, the join-based skyline problem is to find skylines over the joined table $R_1 \bowtie R_2$. To compute the join-based skyline efficiently, several techniques have been proposed [25], [26], [27]. Jin et al. [25] integrate state-of-the-art join methods, such as sort-merge join and nested loop join, with single-relation skyline algorithm. However, it involves a non-trivial processing cost since it needs to access both relations multiple times to generate the correct result. Sun et al. [28] extend the SaLSa [29] algorithm to compute the join-based skyline in a distributed environment and propose an algorithm which prunes the search space iteratively. SFSJ (sort-first-skyline-join) [26] computes the skylines by accessing only a subset of the input tuples. Its main idea is to sort the tuples on each skyline attribute and exploit the property of early termination to determine whether it has accessed sufficient tuples to produce the complete skylines. Instead of performing tuple-to-tuple dominance checks, S^2J (skyline-sensitive join) [27] employs a layer/region pruning strategy. There are some other works aiming to compute the join-based skyline, such as FlexPref [30], SKIN [31], and Prefjoin [32].

Notice that an important pruning principle of the existing algorithms is: tuples that do not belong to group skylines [26] cannot contribute to the join skyline. However, group skylines are very hard to compute in the graph scenario. Furthermore, they do not consider any structural feature to facilitate query process. In contrast, we combine the numeric pruning and the structural pruning based on the data space partitioning.

There are some literatures that try to combine skylines and pattern mining [33], [34], [35]. Papadopoulos et al. [33] define the dominating relation between subgraphs g_i and g_j in G as: g_i dominates g_j if and only if both the connectivity and the number of g_i are no smaller than those of $N(g_j)$, and at least one of the two properties of g_i is strictly larger than that of g_j. The task is to find the SkyGraph, i.e., the set of subgraphs that are not dominated by any other subgraphs. Similarly, Shelokar et al. [35] define the dominating relation utilizing two objectives, support and size of the extracted subgraphs. The idea of skyline queries is integrated into the pattern discovery process to mine skyline patterns [34], which allows users to express the personal preferences easily according to a dominance relation.

Subgraph Search. The subgraph search problem has been extensively studied in the past decades [16], [9]. Ullmann [16] and VF2 [9] are two classic algorithms to verify the subgraph isomorphism between two graphs. In order to improve the efficiency in the subgraph search, most of the proposed algorithms follow filtering-and-verification framework. In the filtering phase, some structural features, including frequent paths [36], trees [37], or subgraphs [38], are chosen as basic index units. A major drawback of these methods is that mining and maintaining the structural features is non-trivial. Therefore, some non-feature-based methods are proposed, such as GCodeing [39], NOVA [40], and SPath [7]. Most of them employ the neighborhood information of vertices, and avoid expensive time and space cost of mining structure features over graphs. Recently, there are emerging researches on querying knowledge graphs [41], [42], [43], [44]. To avoid costly graph isomorphism and edit distance computation, NeMa [41] exploits a neighborhood-based subgraph matching technique for querying real-life networks. GQBE [42] queries knowledge graphs by example entity tuples to improve the usability of knowledge graphs. Mottin et al. introduce exemplar queries and consider the user query to indicate the type of elements that are expected to be in the results [43]. In order to achieve query-specific ranking, Su et al. propose to improve graph queries by relevance feedback [44].

9 Conclusion and Future Work

In this paper, we formalize the problem of subgraph skyline analysis (denoted as S^2A) over large graphs and propose an efficient algorithm to answer S^2A queries. To improve the efficiency, we propose to partition the data space into grid cells, based on which we carefully design feature encoding to facilitate the query process. In order to handle the curse of dimensionality, we propose an attribute cluster-based method. The experimental results on real datasets validate both the effectiveness and efficiency of our method.

As our future work, there are some issues to be addressed. For example, if the number of subgraph skyline answers is still quite large, we can define the *top-k* subgraph skyline as discussed in Section 7.2; Since finding subgraphs that exactly match the queries issued by users is difficult, it is very interesting to redefine the subgraph skyline by considering the structural similarity as a dimension.

References

[1] T. Neumann and G. Weikum, "Rdf-3x: a risc-style engine for rdf," *PVLDB*, vol. 1, no. 1, 2008.

[2] J. Tang, S. Wu, and J. Sun, "Confluence: conformity influence in large social networks," in *KDD*, 2013.

[3] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri, "Querying knowledge graphs by example entity tuples," *CoRR*, vol. abs/1311.2100, 2013.

[4] X. Yu, Y. Sun, P. Zhao, and J. Han, "Query-driven discovery of semantically similar substructures in heterogeneous networks," in *KDD*, 2012.

[5] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao, "gstore: Answering sparql queries via subgraph matching," *PVLDB*, vol. 4, no. 8, 2011.

[6] E. P. F. Chan and H. Lim, "Optimization and evaluation of shortest path queries," *VLDB J.*, vol. 16, no. 3, 2007.

[7] P. Zhao and J. Han, "On graph query optimization in large networks," *PVLDB*, vol. 3, no. 1, 2010.

[8] R. Jin, N. Ruan, S. Dey, and J. X. Yu, "Scarab: scaling reachability computation on large graphs," in *SIGMOD*, 2012.

[9] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. PAMI*, vol. 26, no. 10, 2004.

[10] C. Y. Chan, H. V. Jagadish, K. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-dominant skylines in high dimensional space," in *SIGMOD, Chicago, Illinois, USA, June 27-29*, 2006, pp. 503–514.

[11] Z. Zhang, H. Lu, B. C. Ooi, and A. K. H. Tung, "Understanding the meaning of a shifted sky: a general framework on extending skyline query," *VLDB J.*, vol. 19, no. 2, pp. 181–201, 2010.

[12] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *VLDB*, 2001.

[13] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001.

[14] Y. Tao, X. Xiao, and J. Pei, "Subsky: Efficient computation of skylines in subspaces," in *ICDE*, 2006.

[15] C. Sheng and Y. Tao, "Worst-case i/o-efficient skyline algorithms," *ACM Trans. Database Syst.*, vol. 37, no. 4, 2012.

[16] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, 1976.

[17] W. Zheng, L. Zou, X. Lian, L. Hong, and D. Zhao, "Efficient subgraph skyline search over large graphs," in *CIKM*, 2014, pp. 1529–1538.

[18] S. Yang, Y. Wu, H. Sun, and X. Yan, "Schemaless and structureless graph querying," *PVLDB*, vol. 7, no. 7, pp. 565–576, 2014.

[19] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *VLDB*, 2002.

[20] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, 2005.

[21] W. Jin, A. K. H. Tung, M. Ester, and J. Han, "On efficient processing of subspace skyline queries on high dimensional data," in *SSDBM*, 2007.

[22] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total," in *ICDE*, 1996.

[23] J. Pei, W. Jin, M. Ester, and Y. Tao, "Catching the best views of skyline: A semantic approach based on decisive subspaces," in *VLDB*, 2005.

[24] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang, "Efficient computation of the skyline cube," in *VLDB*, 2005.

[25] W. Jin, M. Ester, Z. Hu, and J. Han, "The multi-relational skyline operator," in *ICDE*, 2007.

[26] A. Vlachou, C. Doulkeridis, and N. Polyzotis, "Skyline query processing over joins," in *SIGMOD*, 2011.

[27] M. Nagendra and K. S. Candan, "Skyline-sensitive joins with lr-pruning," in *EDBT*, 2012.

[28] D. Sun, S. Wu, J. Li, and A. K. H. Tung, "Skyline-join in distributed databases," in *ICDE Workshops*, 2008.

[29] I. Bartolini, P. Ciaccia, and M. Patella, "Salsa: computing the skyline without scanning the whole sky," in *CIKM*, 2006.

[30] J. J. Levandoski, M. F. Mokbel, and M. E. Khalefa, "Flexpref: A framework for extensible preference evaluation in database systems," in *ICDE*, 2010.

[31] V. Raghavan, E. A. Rundensteiner, and S. Srivastava, "Skyline and mapping aware join query evaluation," *Inf. Syst.*, vol. 36, no. 6, 2011.

[32] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Prefjoin: An efficient preference-aware join operator," in *ICDE*, 2011.

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

[34] A. Soulet, C. Raïssi, M. Plantevit, and B. Crémilleux, "Mining dominant patterns in the sky," in *ICDM*, 2011, pp. 655–664.

[35] P. Shelokar, A. Quirin, and O. Cordón, "A multiobjective evolutionary programming framework for graph-based data mining," *Inf. Sci.*, vol. 237, pp. 118–136, 2013.

[36] Y. Tian and J. M. Patel, "Tale: A tool for approximate large graph matching," in *ICDE*, 2008.

[37] S. Zhang, M. Hu, and J. Yang, "Treepi: A novel graph indexing method," in *ICDE*, 2007.

[38] J. Cheng, Y. Ke, W. Ng, and A. Lu, "fg-index: Towards verification-free query processing on graph databases." in *SIGMOD*, 2007.

[39] L. Zou, L. Chen, J. X. Yu, and Y. Lu, "A novel spectral coding in a large graph database," in *EDBT*, 2008.

[40] K. Zhu, Y. Zhang, X. Lin, G. Zhu, and W. Wang, "Nova: A novel and efficient framework for finding subgraph isomorphism mappings in large graphs," in *DASFAA (1)*, 2010.

[41] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan, "Nema: Fast graph search with label similarity," *PVLDB*, vol. 6.

[42] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri, "Querying knowledge graphs by example entity tuples," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 10, pp. 2797–2811, 2015.

[43] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas, "Exemplar queries: Give me an example of what you need," *PVLDB*, vol. 7, no. 5, pp. 365–376, 2014.

[44] Y. Su, S. Yang, H. Sun, M. Srivatsa, S. Kase, M. Vanni, and X. Yan, "Exploiting relevance feedback in knowledge graph search," in *SIGKDD*, 2015, pp. 1135–1144.

[45] U. Feige, V. S. Mirrokni, and J. Vondrák, "Maximizing non-monotone submodular functions," in *FOCS, Providence, RI, USA*, 2007, pp. 461–471.

[46] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "Np-hardness of euclidean sum-of-squares clustering," *Machine Learning*, vol. 75, no. 2, pp. 245–248, 2009.

Weiguo Zheng received his B.E. degree in School of Computer Science and Technology from China University of Mining and Technology (CUMT), in 2010, and the Ph.D. degree in computer science from Peking University. He is now a postdoctoral research fellow in the Chinese University of Hong Kong, focusing on graph database management.

Xiang Lian received the B.S. degree from Nanjing University in 2003, and the Ph.D. degree in computer science from the Hong Kong University of Science and Technology. He is now an assistant professor in the Department of Computer Science at the University of Texas Rio Grande Valley. His research interests include probabilistic data management and probabilistic RDF graphs.

Lei Zou received his B.S. degree and Ph.D. degree in Computer Science at Huazhong University of Science and Technology (HUST) in 2003 and 2009, respectively. Now, he is an associate professor in Institute of Computer Science and Technology of Peking University. His research interests include graph database and semantic data management.

Liang Hong received his BS degree and Ph.D. degree in Computer Science at Huazhong University of Science and Technology (HUST) in 2003 and 2009, respectively. Now, he is an associate professor in School of Information Management of Wuhan University. His research interests include graph database, spatio-temporal data management and social networks.

Dongyan Zhao received the B.S. degree, M.S. degree and Ph.D. degree from Peking University in 1991, 1994 and 2000, respectively. Now, he is a professor in Institute of Computer Science and Technology of Peking University. His research interest is on information processing and knowledge management, including computer network, graph database, and intelligent agent.