Influential Community Search over Large Heterogeneous Information Networks

Yingli Zhou
The Chinese University of Hong Kong, Shenzhen
yinglizhou@link.cuhk.edu.cn

Yixiang Fang*
The Chinese University of Hong Kong, Shenzhen
fangyixiang@cuhk.edu.cn

Wensheng Luo
The Chinese University of Hong Kong, Shenzhen
luowensheng@cuhk.edu.cn

Yunming Ye
Harbin Institute of Technology, Shenzhen
yeyunming@hit.edu.cn

ABSTRACT

Recently, the topic of influential community search has gained much attention. Given a graph, it aims to find communities of vertices with high importance values from it. Existing works mainly focus on conventional homogeneous networks, where vertices are of the same type. Thus, they cannot be applied to heterogeneous information networks (HINs) like bibliographic networks and knowledge graphs, where vertices are of multiple types and their importance values are of heterogeneity (i.e., for vertices of different types, their importance meanings are also different). In this paper, we study the problem of influential community search over large HINs. We introduce a novel community model, called heterogeneous influential community (HIC), or a set of closely connected vertices that are of the same type and high importance values, using the meta-path-based core model. An HIC not only captures the importance of vertices in a community, but also considers the influence on meta-paths connecting them. To search the HICs, we mainly consider meta-paths with two and three vertex types. Then, we develop basic algorithms by iteratively peeling vertices with low importance values, and further propose advanced algorithms by identifying the key vertices and designing pruning strategies that allow us to quickly eliminate vertices with low importance values. Extensive experiments on four real large HINs show that our solutions are effective for searching HICs, and the advanced algorithms significantly outperform baselines.

PVLDB Reference Format:
Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. Influential Community Search over Large Heterogeneous Information Networks. PVLDB, 16(8): 2047 - 2060, 2023.
doi:10.14778/3594512.3594532

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/JayLZhou/ICSH.

1 INTRODUCTION

Heterogeneous information networks (HINs) are networks with multiple typed objects and multiple typed links denoting different semantic relations. These graph data sources are prevalent in various domains, including bibliographic networks [40, 46], IMDB movie networks [43, 57], and knowledge graphs [15]. The objects of these HINs are often associated with importance values, which can be found easily from their profiles or derived by importance estimation methods [19, 20, 36, 37]. ***Particularly, the importance values are also of heterogeneity; that is, for vertices of different types, their importance semantic meanings are different, and their possible values are in different ranges.*** For example, the HIN of DBLP network has four vertex types, i.e., author (A), paper (P), venue (V), and topic (T); the objects' importance values can be reflected by their h-indexes, citation numbers, h5-indexes, and popularity values (i.e., the numbers of results in Google search engine), respectively. In Figure 1, the example HIN of DBLP network consists of seven authors (i.e., $a_1, \cdots, a_7$), five papers (i.e., $p_1, \cdots, p_5$), one venue (i.e., v_1), and one topic (i.e., t_1). The directed lines denote their semantic relationships. For instance, the author a_3 has *written* a paper p_3, which *mentions* the topic t_1, *published* in the venue v_1. The vertices' importance values are shown in the HIN (e.g., the author a_3's h-index is 60, and the paper p_4's citation number is 400). Note that the h-index of an author is often hundreds at most [35], while a paper could be cited thousands of times or more [52].

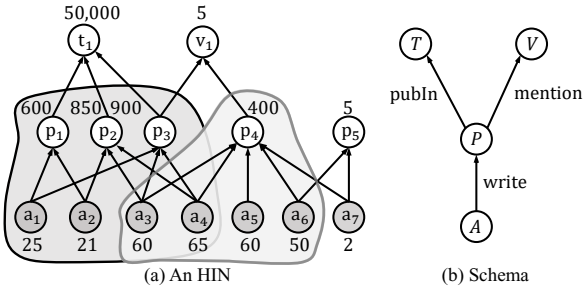


Figure 1: An example HIN of DBLP network.

In this paper, we study the problem of **I**nfluential **C**ommunity **S**earch over **HI**Ns (or ICSH problem). The topic of influential CS , or finding communities of vertices with high influence values from a graph, has gained much attention recently [2, 8, 26–29, 31, 53]. However, existing works mainly focus on conventional homogeneous networks where vertices are of the same type, making them inapplicable for the HIN which involves multiple types of vertices and edges. To tackle this issue, we aim to search highly influential communities from HINs. A highly influential community is a set of vertices of the same type, which are not only closely related, but also have high importance values. Particularly, the community satisfies the *meta-path*-based cohesiveness [15], i.e., its vertices are intensively connected by instances of a specific meta-path [46] or

*Yixiang Fang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 16, No. 8 ISSN 2150-8097.
doi:10.14778/3594512.3594532

Table 1: Existing works of community search (CS).

Graph type	Non-influential CS	Influential CS
Homogeneous	e.g., [10, 11, 13, 21, 23, 44]	[2, 8, 22, 26–29, 31, 38, 53]
Heterogeneous	e.g., [14, 15, 24, 25, 39]	**ICSH** (this work)

a path of vertex types and edge types between two given vertex types. In Figure 1, the meta-path $\mathcal{P}_1$=(APA), defined on authors (A) and papers (P), describes two authors with co-authorship. The authors $\{a_3, a_4, a_5, a_6\}$ form a cohesive community, since each pair of authors can be connected by a path instance of $\mathcal{P}$, and the authors and papers are of high h-indexes and citation numbers respectively.

Prior works. Existing works on community retrieval can be roughly classified into *community detection* and *community search* (CS). Community detection often aims to identify all communities for a graph [16, 32, 34, 47–49, 62], so they are not "query-based" (i.e., they are not customized for a query request) or efficient for processing large graphs. To tackle these limitations, the CS solutions (e.g., [11, 13, 21, 44]) have been extensively studied in the last decade, which can be classified as shown in Table 1. Most of the existing CS solutions focus on the homogeneous graph where all the vertices are of the same type, and they can be further classified as non-influential CS and influential CS. In particular, influential CS [2, 8, 26, 27, 29, 31, 53] aims to find communities of vertices with high importance values. However, these CS solutions cannot be applied to the HIN because it involves multiple types of vertices and edges with different semantic meanings and it does not make sense to mix them up for performing CS using previous solutions. Recently, the topic of CS over HINs has received much interest [14, 15, 24, 25]. Nevertheless, none of the them has studied how to perform influential CS over HINs by considering vertices' importance values.

ICSH problem. Our ICSH problem aims to search highly influential communities from HINs. A highly influential community is a set of vertices with the same type, which are not only closely related, but also have high importance values. To achieve this goal, we face two key questions: (1) How to model the cohesiveness of vertices with the same type in a community? (2) How to model the influential community by considering the vertices' importance values? For the first question, the existing CS over HINs query [15] performs CS over HINs, by introducing the $(k, \mathcal{P})$-core, or the set of vertices in which each vertex has at least k neighbors linked by the path instances of meta-path $\mathcal{P}$. The $(k, \mathcal{P})$-core is effective for modeling the cohesiveness of vertices with the same type in a community, since a meta-path with a limited length between two vertices well reveals their semantic relationships (e.g., $\mathcal{P}_1$=(APA) indicates the co-authorship between two authors).

For the second question, to model a highly influential community, a typical group of existing influential CS works [2, 8, 26–29, 31, 53] maximizes the minimum importance value of all vertices that form a connected k-core, or a cohesive subgraph such that each vertex has at least k neighbors within it [13]. However, since they inherently assume that all the vertices are of the same type, meaning that their importance should be of the same semantic meanings (but may have different values), they cannot be directly applied to the HIN, due to the heterogeneity of importance values of vertices, i.e., for vertices of different types, their importance semantic meanings are different and their values are often in different ranges.

Another more related group of existing influential CS works [26, 28] focuses on the multi-valued network with each vertex having multiple importance values, represented by a multi-dimensional vector. Instead of maximizing a single minimum value of all vertices that form a k-core, it introduces an influence vector for a community, by using the minimum importance values of each dimension. Then, it finds all the communities with skyline influence vectors, or vectors that cannot be dominated by the influence vectors of all other possible communities. Note that a vector v_1 is said to be dominated by another one v_2, if for each dimension, the value of v_1 is less or equal to that of v_2, but for a certain dimension, it is strictly less than that of v_2. In this paper, we exploit this idea for modeling the importance values of influential communities of HINs. Specifically, we consider the induced sub-HIN of $(k, \mathcal{P})$-core, and for each vertex type in the meta-path $\mathcal{P}$, we consider the minimum importance value of all the vertices with that type. As a result, by considering all the vertex types in $\mathcal{P}$, we can get an influence vector for a community, and then find all the communities with skyline influence vectors that cannot be dominated by each other.

By combining the answers of the two questions above, we introduce a novel community model, called heterogeneous influential community (HIC), which is a set of vertices in a meta-path-based core and its induced subgraph has the skyline influence vector. An HIC not only captures the importance value of vertices in a community, but also considers the influence on meta-paths connecting them. In Figure 1, for example, consider the meta-path $\mathcal{P}_1$=(APA). There are two communities of authors, whose induced sub-HINs include vertex sets $\{a_1, a_2, a_3, a_4, p_1, p_2, p_3\}$ and $\{a_3, a_4, a_5, a_6, p_4\}$, and their skyline influence vectors are (21, 600) and (50, 400) respectively. Note that if a CS over HINs query is issued with k=3 and $\mathcal{P}_1$, then the returned community will contain all the authors with influence vector (2, 5), which is clearly dominated by those of our ICSH communities.

Applications. The HIC can be used in various real applications, such as event organization, recommendation, and network analysis. For example, an academic workshop is often organized by some senior researchers with close relationships, so to organize a workshop, we can invite members of the influential communities of researchers from DBLP network, who have not only close relationships but also high h-indexes. In Twitter, as another example, the relationship between users and topics can be modeled as an HIN. By finding the influential communities from it, we can identify some prestige users and hot topics, and thus some recommendation tasks based on these users and topics can be performed.

Our technical contributions. To find all the HICs from an HIN, we first focus on the case that the meta-path $\mathcal{P}$ has only two vertex types and present a basic algorithm, which computes the skyline influence vectors by iteratively finding the maximum importance value for each vertex type. Although this algorithm is straightforward and also easy to implement, it is very costly since it peels vertices one by one and involves much redundant computation. To improve the efficiency, we propose an advanced algorithm which avoids the removal operations and unnecessary search by introducing a novel concept, namely target-keynode, extended from the concept of keynode for influential CS on homogeneous graphs [2]. Moreover, we observe that for each target-keynode, an upper bound value exists for its corresponding influence vector, and it is very close to the actual value. Hence, we further boost the efficiency by exploiting the upper bound value.

For the meta-paths with three vertex types, we first propose a basic algorithm using the idea of dimension reduction. In specific,

we first find all possible importance values in the third vertex type, and then by fixing each value of them, we invoke the advanced algorithm above to compute the 2-dimensional influence vector. This algorithm, however, may involve much invalid search and find communities with influence vectors dominated by others. To tackle this issue, we introduce an advanced algorithm, which significantly reduces the number of influence vectors dominated by others by an effective pruning strategy. Note that in this paper, we mainly focus on meta-path with lengths at most four because long meta-paths are often meaningless and rarely used in practice [15, 25], but our algorithms above can be easily extended for processing them.

Extensive experimental evaluation on four real large HINs shows that our ICSH solutions are effective for finding the influential communities. Besides, our advanced algorithms are up to two orders of magnitude faster than the basic algorithms for meta-paths with two and three vertex types, respectively.

Outline. We formulate our ICSH problem in Section 2. Sections 3 and 4 present the ICSH algorithms for processing meta-paths with two and three vertex types, respectively. We report the experimental results in Section 5. We review the related works in Section 6 and conclude in Section 7.

2 PROBLEM FORMULATION

In this section, we first review some basic concepts of HINs, and then formally formulate our ICSH problem.

2.1 Preliminaries

Definition 1. ***HIN.*** *An HIN is a directed graph $\mathcal{H} = (V, E, \psi, \phi, \omega)$ with a vertex type mapping function $\psi : V \rightarrow \mathcal{A}$, an edge type mapping function $\phi : E \rightarrow \mathcal{R}$, and an importance function $\omega : V \rightarrow \mathbb{R}$, where each vertex $v \in V$ belongs to a vertex type $\psi(v) \in \mathcal{A}$ and has an importance value $\omega(v)$, and each edge $e \in E$ belongs to an edge type $\phi(e) \in \mathcal{R}$, and $|\mathcal{A}| + |\mathcal{R}| > 2$.*

Note that the above HIN model includes some special kinds of graphs, such as bipartite graphs, k-partite graphs, and multigraphs where multiple edges may exist between the same pair of vertices. An HIN often follows a schema, which is a directed graph defined over vertex types $\mathcal{A}$ and edge types $\mathcal{R}$, denoted by $T_G = (\mathcal{A}, \mathcal{R})$. As shown in the DBLP network schema of Figure 1(b), the schema describes all allowable edge types between vertex types. Note that in the schema, if there is an edge R from vertex type A to vertex type B, the inverse edge R^{-1} naturally exists from B to A.

Definition 2. ***Meta-path [46]*** *A meta-path $\mathcal{P}$ is a path defined on an HIN schema $T_G = (\mathcal{A}, \mathcal{R})$, and is denoted in the form $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$, where l is the length of $\mathcal{P}$, $A_i \in \mathcal{A}$, and $R_i \in \mathcal{R} (1 \leq i \leq l)$.*

If there exist no multiple edges between the same pair of vertex types, we simply use vertex type names to denote a meta-path, i.e., $\mathcal{P} = (A_1 A_2 \cdots A_{l+1})$. The reverse meta-path of $\mathcal{P}$ denoted by $\mathcal{P}^{-1}$ is the reverse path of $\mathcal{P}$ in T_G. A meta-path $\mathcal{P}$ is symmetric if it is the same with $\mathcal{P}^{-1}$. Clearly, the meta-path $\mathcal{P}_1$=(APA) on the HIN of DBLP network is a symmetric meta-path.

Given an HIN $\mathcal{H}$, a path p=$(a_1 \rightarrow a_2 \cdots \rightarrow a_{l+1})$ between vertices $a_1, a_{l+1} \in \mathcal{H}$ is called a path instance of $\mathcal{P}$, if $\forall 1 \leq i \leq l$, the vertex a_i and edge $e_i = (a_i, a_{i+1})$ satisfy $\psi(a_i) = A_i$ and $\phi(e_i) = R_i$. For example, in Figure 1, the path $a_1 \rightarrow p_1 \rightarrow a_2$ is a path instance of $\mathcal{P}_1$. Note that we use lowercase letters (e.g., a_1) to denote vertices in an HIN, and upper-case letters (e.g., A) to denote vertex types. We say that vertex u is a $\mathcal{P}$-neighbor of vertex v, or (u, v) is a $\mathcal{P}$-pair, if they are connected by an instance of $\mathcal{P}$. Two vertices u and v are $\mathcal{P}$-connected if there exists a chain of vertices from u to v such that any two adjacent vertices are $\mathcal{P}$-neighbors.

Definition 3. ***$\mathcal{P}$-Graph.*** *Given an HIN $\mathcal{H}$ and a symmetric meta-path $\mathcal{P}$, the $\mathcal{P}$-graph is a homogeneous graph $\mathcal{H}_\mathcal{P}$ such that each $\mathcal{P}$-pair of $\mathcal{H}$ corresponds to an edge in $\mathcal{H}_\mathcal{P}$.*

Next, we present the concept of $(k, \mathcal{P})$-core, a widely-used cohesive subgraph model on HINs [15, 25, 54]. Here, the symmetric meta-path $\mathcal{P}$=$(A_1 A_2 \cdots A_h \cdots A_2 A_1)$ has h vertex types, and the type linked by $\mathcal{P}$, A_1, is called the *target type*. Given a vertex v and a set S of vertices with target type, the number of $\mathcal{P}$-neighbors of v within S is called $\mathcal{P}$-degree, denoted by $\alpha(v, S)$.

Definition 4. ***$(k, \mathcal{P})$-core [15].*** *Given an HIN $\mathcal{H}$, an integer k, and a symmetric meta-path $\mathcal{P}$, the $(k, \mathcal{P})$-core of $\mathcal{H}$ is a set $S_{k,\mathcal{P}}$ of $\mathcal{P}$-connected vertices, s.t. $\forall v \in S_{k,\mathcal{P}}$, $\alpha(v, S_{k,\mathcal{P}}) \geq k$, where vertices in $S_{k,\mathcal{P}}$ are with the target type.*

Table 2 summarizes the notations frequently used in this paper.

Table 2: Notations and meanings.

Notation	Meaning
$\mathcal{H} = (V, E, \psi, \phi, \omega)$	An HIN with vertex set V, edge set E, vertex type mapping function ψ, edge type mapping function ϕ, and vertex importance function ω
$\mathcal{P}$	A symmetric meta-path defined on the schema of $\mathcal{H}$
Φ	A set of path instances of a meta-path $\mathcal{P}$
$\mathcal{H}[\Phi]$	A sub-HIN induced by path instances in Φ
$\alpha(v, S)$	$\mathcal{P}$-degree, i.e., the number of path instances starting from the vertex v and ending at vertices in a set S
h	The number of vertex types in $\mathcal{P}$
$\mathcal{K}$	A list of target-keynodes in $\mathcal{H}$
θ_i	The smallest importance value in the vertex set V_i

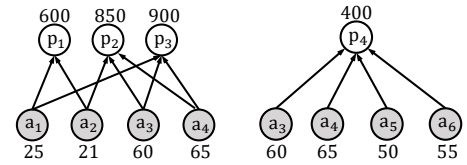
2.2 Problem statement

We first introduce a novel concept of $\mathcal{P}$-induced sub-HIN.

Definition 5. ***$\mathcal{P}$-induced sub-HIN.*** *Given an HIN $\mathcal{H}$ and a set Φ of path instances of a meta-path $\mathcal{P}$, the $\mathcal{P}$-induced sub-HIN of Φ, denoted by $\mathcal{H}[\Phi]$, is a subgraph of $\mathcal{H}$ containing the union of all vertices and union of all edges that are in the path in Φ.*

As aforementioned, our proposed HIC considers not only vertices with target type, but also meta-paths connecting them. As a result, an HIC corresponds to a $\mathcal{P}$-induced sub-HIN. To model the influence of an HIC, we consider the importance values of all the vertices in the HIC. However, due to the heterogeneity of importance values in the HIN, for vertices with different types, their importance values are not comparable. Besides, some important target vertices may be connected by intermediate vertices with low importance values, which may make the relationship less important. To solve this issue, we propose to use a vector of multiple dimensions to characterize the influence of an HIC, such that each dimension captures the importance of a specific vertex type.

Definition 6. ***Influence vector.*** *Given an HIN $\mathcal{H}$=$(V, E, \psi, \phi, \omega)$, a symmetric meta-path $\mathcal{P}$=$(A_1 A_2 \cdots A_h \cdots A_2 A_1)$, and a $\mathcal{P}$-induced*



(a) $\mathcal{G}$ and $f(\mathcal{G})$=(21, 600) (b) $\mathcal{G}'$ and $f(\mathcal{G}')$=(50, 400)

Figure 2: Two $\mathcal{P}$-induced sub-HINs ($\mathcal{G}$ and $\mathcal{G}'$) in Figure 1.

sub-HIN $\mathcal{G}$=($V_\mathcal{G}$, $E_\mathcal{G}$, ψ, ϕ, ω), the influence vector of $\mathcal{G}$ is

$$f(\mathcal{G}) = (min_{v \in V_1} \omega(v), min_{v \in V_2} \omega(v), \cdots, min_{v \in V_h} \omega(v)), \quad (1)$$

where $V_i \subset V_\mathcal{G}$ is the set of vertices whose types are the i-th vertex type of $\mathcal{P}$, i.e., $\forall v \in V_i$, $\psi(v)$=A_i.

Here, in line with most of the existing works of influential CS [2, 8, 26–29, 31, 53], we adopt the $min(\cdot)$ function to measure the importance for each vertex type. The main reason is that it requires each vertex of an HIC to have a large importance value, indicating that every member is an influential individual [27].

Other functions such as $max(\cdot)$, $sum(\cdot)$, and $avg(\cdot)$ can also be used [38]. For $max(\cdot)$, our algorithms can be used directly by changing the importance values of all vertices by multiplying -1. For $sum(\cdot)$ and $avg(\cdot)$, we will study how to solve these cases in the future.

Definition 7. ***Dominance relationship of influence vectors.*** *Given two influence vectors $f(\mathcal{G})$ and $f(\mathcal{G}')$ with the same dimension h, we say $f(\mathcal{G}')$ dominates $f(\mathcal{G})$, if for any $i \in \{1, 2, \cdots, h\}$,*

$$f_i(\mathcal{G}) \leq f_i(\mathcal{G}'), \quad (2)$$

and there exists a certain i s.t.

$$f_i(\mathcal{G}) < f_i(\mathcal{G}'), \quad (3)$$

where $f_i(\mathcal{G})$ is the i-th element of the vector $f(\mathcal{G})$.

For a set of influence vectors with the same dimension, if one influence vector cannot be dominated by any other vectors, then it is called a *skyline influence vector*. Next, we formally introduce the concept of *heterogeneous influential community (HIC)*.

Definition 8. ***HIC.*** *Given an HIN $\mathcal{H}$, a positive integer k, and a symmetric meta-path $\mathcal{P}$, an HIC is a set S of vertices with the target type, if there exists a set Φ of path instances of $\mathcal{P}$ between them satisfying properties:*

*1. **Link cohesiveness**. Vertices of S can form a $(k, \mathcal{P})$-core using Φ, i.e., $\forall v \in S$, it has at least k $\mathcal{P}$-neighbors in S via paths in Φ;*

*2. **Skyline importance**. There is no other sets S' and Φ', s.t. they satisfy Property 1 and $f(\mathcal{H}[\Phi'])$ dominates $f(\mathcal{H}[\Phi])$;*

*3. **Maximality**. There is no other set S' of vertices with target type, satisfying $S \subset S'$ and the two properties above.*

Essentially, an HIC is a $(k, \mathcal{P})$-core with vertices of high importance values in its $\mathcal{P}$-induced sub-HIN.

Problem 1 (ICSH problem). *Given an HIN $\mathcal{H}$, a positive integer k, and a symmetric meta-path $\mathcal{P}$, return all the HICs from $\mathcal{H}$.*

Example 1. *In Figure 1, let $\mathcal{P}_1$=(APA) and k=3. Then, we will find two HICs $\{a_1, a_2, a_3, a_4\}$ and $\{a_3, a_4, a_5, a_6\}$ in the HIN. Figures 2(a) and 2(b) show two $\mathcal{P}$-induced sub-HINs, $\mathcal{G}$ and $\mathcal{G}'$, corresponding to these two HICs respectively. We can easily verify that $f(\mathcal{G})$=(21, 600) and $f(\mathcal{G}')$=(50, 400) are the skyline influence vectors.*

In practice, people often use meta-paths with limited lengths since they are more meaningful than long meta-paths [15, 25, 46]. Thus, we focus on finding HICs for meta-paths with lengths less than four, so the meta-paths either have two or three vertex types. Nonetheless, our algorithms can be easily extended to process meta-paths with more than two vertex types, as we will show later.

3 ALGORITHMS FOR THE CASE h=2

In this section, we present two algorithms for solving the ICSH problem with meta-paths having two vertex types (i.e., h=2), which means that the influence vector of an HIC is a 2-dimensional vector f=(f_1, f_2). For ease of exposition, we denote the two sets of vertices with the target type and the other type in the input HIN by V_1 and V_2, respectively.

We use θ_1 and θ_2 to denote the smallest importance values in V_1 and V_2 respectively. For lack of space, all the proofs in this paper are included in Appendix D of the technical report [61].

3.1 A basic algorithm for the case h=2

To find all the HICs with skyline influence vector $f = (f_1, f_2)$, our core idea is to iteratively fix the value of one dimension of f, and in the meantime maximize the value of the other one. Specifically, in each iteration, we perform three steps: (1) fix the value of f_1 and maximize the value of f_2 by finding a $(k, \mathcal{P})$-core; (2) fix the value of f_2 and maximize the value of f_1 by finding a $(k, \mathcal{P})$-core such that vertices of the other type have importance values being at least f_2; and (3) record a skyline influence vector and its corresponding HIC. In the next iteration, we first increase f_1 to the minimum value that is larger than the previous f_1, and then repeat the three steps above. This process is repeated until we cannot increase f_1. Since the value of f_1 is increased in these iterations, we can keep peeling vertices in V_1 with the lowest importance values.

Algorithm 1: `Basic2D`($\mathcal{H}, k, \mathcal{P}$)

input : An HIN $\mathcal{H}$, an integer k, and a meta-path $\mathcal{P}$
output: All HICs and their skyline influence vectors

$f_1 \leftarrow \theta_1, f_2 \leftarrow \infty, \mathcal{R} \leftarrow \emptyset$;
while $f_2 > \theta_2$ **do**
  $f_2 \leftarrow$ `TypeMax`($\mathcal{H}, f_1, \theta_2, V_2$);
  $f_1 \leftarrow$ `TypeMax`($\mathcal{H}, f_1, f_2, V_1$);
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{(f_1, f_2)\}$;
  $f_1 \leftarrow \min\{\omega(v_1) | v_1 \in V_1 \wedge \omega(v_1) > f_1\}$;
return $\mathcal{R}$ and the corresponding HICs;

Function `TypeMax`($\mathcal{H}, f_1, f_2, V$):
  remove vertices $\{v_1 | (v_1 \in V_1 \wedge \omega(v_1) < f_1)\}$ and $\{v_2 | (v_2 \in V_2 \wedge \omega(v_2) < f_2)\}$ from $\mathcal{H}$;
  $S_1 \leftarrow$ compute a $(k, \mathcal{P})$-core from $\mathcal{H}$;
  $S_2 \leftarrow$ all the vertices in V_2 that are in the path instances of $\mathcal{P}$ between vertices in S_1;
  $f \leftarrow \min\{\omega(v) | v \in V\}$;
  while $V \cap (S_1 \cup S_2) \neq \emptyset$ **do**
    $v \leftarrow \text{argmin}_{v \in V} \omega(v)$;
    $f \leftarrow \omega(v)$;
    `DeleteVertex`($\mathcal{H}, S_1, S_2, v$);
  return f;

Based on the idea above, we develop an algorithm, called `Basic2D`, as shown in Algorithm 1. Specifically, we first initialize f_1=θ_1, f_2=∞,

and $\mathcal{R}$=$\emptyset$ which is used to keep skyline influence vectors (line 1). Then, we use a while loop to find all the skyline influence vectors with corresponding HICs (lines 2-6). In each iteration, we first fix f_1 and maximize the value of f_2 by invoking TypeMax (line 3), then fix f_2 and maximize the value of f_1 by calling TypeMax (line 4), followed by recording the skyline influence vector (line 5), and finally increase f_1 for the next iteration (line 6).

In Algorithm 1, TypeMax is a function for computing the maximum importance value of a certain vertex type. It first removes all vertices with importance values less than f_1 and f_2 from V_1 and V_2 respectively (line 9). Then, it computes a $(k, \mathcal{P})$-core S_1, and collects the vertices in V_2 that are in the path instances of $\mathcal{P}$ between vertices in S_1 (lines 10-11). Finally, it iteratively removes vertices with the smallest importance values by invoking function DeleteVertex (please see Appendix A of technical report [61]), until no more vertices can be removed when an HIC with a skyline importance vector is obtained (lines 13-16).

Theorem 3.1. *Algorithm 1 correctly computes all the HICs with 2-dimensional skyline influence vectors.*

Example 2. *Consider Figure 1 with k=3 and $\mathcal{P}$=(APA). We first compute the maximum value of f_2 by* Typemax$(\mathcal{H}, 2, 5, V_1)$ *which removes p_5 and p_4, and then get f_2=$\omega(p_1)$=600. Next, we compute the maximum value of f_1 by fixing f_2=600 and invoking* TypeMax$(\mathcal{H}$, $2, 600, V_1)$ *(note that if a_7 and a_2 are removed, there is no $(3, \mathcal{P})$-core), and get f_1=21. After obtaining the first skyline influence vector (21, 600), we increase f_1 to the minimum value that is larger than 21, which is 25. By repeating the above process, we can get the next skyline influence vector (50, 400). After a_1, a_2, a_6, and a_7 are removed, there is no $(3, \mathcal{P})$-core in the HIN, and f_2 is set to 5, so we stop. Figure 2 depicts the two HICs and their corresponding $\mathcal{P}$-induced sub-HINs.*

Lemma 3.2. *The total time cost of* Basic2D *is $O(s \cdot n_2 \cdot (n_1 + m) + n_1 \cdot b_{1,2} + n_1 \cdot n_2 \cdot b_{2,1})$, where n_i=$|V_i|$, $b_{i,j}$ is the maximum number of vertices in V_i that are connected to a vertex in V_j, m is the number of edges in the $\mathcal{P}$-graph and s is the number of skyline influence vectors.*

3.2 An advanced algorithm for the case h=2

Although Basic2D is straightforward and also easy to implement, it is very costly because it removes vertices from V_2 on the entire HIN multiple times, leading to too much redundant computation. To alleviate this issue, we propose a novel algorithm to eliminate redundant computation and reduce peel operations based on the definition of target-keynode, which is extended from the concept of keynode used for influential CS on homogeneous graphs [2].

Definition 9. ***Target-keynode.*** *Given an HIN $\mathcal{H}$, an integer k, and a meta-path $\mathcal{P}$, a vertex u with target type in $\mathcal{H}$ is a target-keynode, if there exists a $(k, \mathcal{P})$-core such that the minimum importance value of its vertices is $\omega(u)$.*

Essentially, each target-keynode corresponds to a non-empty $(k, \mathcal{P})$-core, but this $(k, \mathcal{P})$-core may not be an HIC. The main idea of our algorithm is to first obtain all the target-keynodes. Then, we find a list of possible skyline influence vectors f=(f_1, f_2) by exploiting these target-keynodes. Specifically, for each target-keynode u, we first fix f_1=$\omega(u)$, then compute the upper bound of f_2, denoted by $\widehat{f_2}(u)$, and finally shrink $\widehat{f_2}(u)$ to its actual value. Afterwards, the skyline influence vectors with HICs can be found.

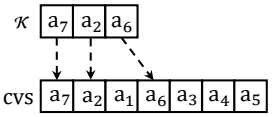


Figure 3: A running example of ComputeTK**.**

3.2.1 Obtaining target-keynodes. All the target-keynodes can be obtained by iteratively peeling vertices with the smallest importance values in the $(k, \mathcal{P})$-core. Specifically, we first obtain the $(k, \mathcal{P})$-core S_1 from the HIN and a $\mathcal{P}$-induced sub-HIN $\mathcal{H}[\Phi]$, where Φ is a set of path instances of $\mathcal{P}$ between vertices in the S_1. Then, we remove the vertex u with the smallest importance value in S_1, and append it to a list $\mathcal{K}$ as a target-keynode. After u is removed from S_1, all the vertices with $\mathcal{P}$-degrees less than k are iteratively removed and appended to *cvs* together with u , where *cvs* is a vertex set for subsequent computation. To obtain all the target-keynodes, we repeat the above process until S_1 becomes empty.

Algorithm 2: ComputeTK($\mathcal{H}$, k, $\mathcal{P}$)

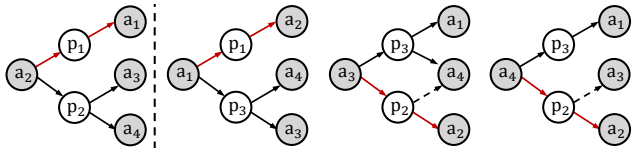
```
1  K ← ∅, cvs ← ∅, Q ← ∅;
2  S1 ← compute the (k, P)-core in H;
3  H[Φ] ← a P-induced sub-HIN of a set of path instances of P
     between vertices in S1 ;
4  while S1 ≠ ∅ do
5      u ← argmin_{v∈S1}{ω(v)}; Q.add(u) ;
6      append u to the end of K ;
7      while Q ≠ ∅ do
8          v ← Q.poll() ;
9          foreach v' ∈ N(v, H[Φ]) do
10             if α(v', S1) = k then Q.add(v') ;
11             α(v', S1) ← α(v', S1) - 1;
12         delete v from S1 and H[Φ]; append v to the end of cvs;
13 return K, cvs, H[Φ];
```

Algorithm 2 shows the steps above. We first initialize $\mathcal{K}$=$\emptyset$, *cvs*=$\emptyset$, and Q is an auxiliary queue. Next, we compute the $(k, \mathcal{P})$-core S_1 and a $\mathcal{P}$-induced sub-HIN $\mathcal{H}[\Phi]$ of a set of path instances between vertices in S_1 (lines 2-3). Then, we remove the vertex with the smallest importance value in S_1 from $\mathcal{H}[\Phi]$, and place it into $\mathcal{K}$ as a target-keynode (lines 5-6). The remaining vertices with $\mathcal{P}$-degrees less than k are also removed and appended to *cvs* (lines 8-12). Finally, when S_1=$\emptyset$, we return $\mathcal{K}$, *cvs*, and $\mathcal{H}[\Phi]$ (line 13).

Example 3. *In Figure 1, $\mathcal{P}_1$=(APA) and k=3. To compute all target-keynodes, we first obtain the (3, $\mathcal{P}$)-core S_1=$\{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$ from the HIN and a $\mathcal{P}$-induced sub-HIN $\mathcal{H}[\Phi]$=$\{a_1, a_2, a_3, a_4, a_5, a_6, a_7, p_1, p_2, p_3, p_4, p_5\}$. We first obtain a target-keynode a_7 and append it to $\mathcal{K}$ and cvs. Then, we remove a_2 from $\mathcal{H}[\Phi]$ and append it to $\mathcal{K}$ as a target-keynode. After that, a_1 is iteratively removed and appended to cvs together with a_2 immediately. Similarly, we can obtain another target-keynode a_6, and a_6, a_3, a_4 and a_5 are recorded into cvs. After removing a_7, a_2, and a_6, all the other vertices in S_1 are also removed, so we stop. Figure 3 shows the process above.*

3.2.2 Computing $\widehat{f_2}(u)$ for each target-keynode u. We begin with an interesting observation about target-keynode.

Observation 1. *A target-keynode must have at least k $\mathcal{P}$-neighbors whose $\mathcal{P}$-degrees are at least k.*



(a) compute $ub(a_2)$ (b) compute $ub(a_2, S_1)$

Figure 4: An example of computing $\widehat{f_2}(a_2)$.

To establish the relationship between $\widehat{f_2}(u)$ of a target-keynode u and the importance values of vertices in V_2, we introduce the following definition.

Definition 10. ***Skyline path.*** *Given an HIN $\mathcal{H}$, a meta-path $\mathcal{P}$, and a $\mathcal{P}$-pair (u, v), a path instance of $\mathcal{P}$ p=$u \rightarrow a_1 \cdots \rightarrow a_n \rightarrow v$ is the skyline path of (u, v), if there is no other path instance of $\mathcal{P}$ p'=$u \rightarrow a'_1 \cdots \rightarrow a'_n \rightarrow v$ from u to v s.t. $\forall i \in \{1, 2, \cdots, n\}$, $\omega(a_i) \leq \omega(a'_i)$ and $\exists i \in \{1, 2, \cdots, n\}, \omega(a_i) < \omega(a'_i)$.*

In the case h=2, the skyline path of a $\mathcal{P}$-pair (u, v) is the path instance with the largest importance value of the middle vertex among all path instances between u and v, and we call this value the *weight of this $\mathcal{P}$-pair*. Obviously, for each vertex u in S_1, it has $\alpha(u, S_1)$ skyline paths, where $\alpha(u, S_1)$ is the $\mathcal{P}$-degree of u and S_1 is the corresponding $(k, \mathcal{P})$-core containing it.

According to Observation 1, since the $\mathcal{P}$-degree of target-keynode u is at least k, $\widehat{f_2}(u)$ is bounded by the k-th largest value of the weights of all the $\mathcal{P}$-pair containing u, which we denote as $ub(u)$. Besides, for each $\mathcal{P}$-neighbor v_i of u, we also find the k-th largest $\mathcal{P}$-pair's weight of v_i and put it into the set F, i.e., F=$\{ub(u, v_i)|v_i \in N(u, S_1)\}$. Since there are at least k vertices in $N(u, S_1)$ with $\mathcal{P}$-degree being at least k, we take the k-th largest value in F as an upper bound of f_2, and denote it as $ub(u, N(u, S_1))$. Finally, we set the upper bound $\widehat{f_2}(u)$ as

$$\widehat{f_2}(u) = \min\{ub(u),\ ub(u, N(u, S_1))\}. \tag{4}$$

We further illustrate this process by Example 4.

Example 4. *In Example 3, a_7, a_2, and a_6 are target-keynodes. To compute $\widehat{f_2}(a_2)$, we first identify skyline paths by removing the edges on other path instances, as shown in Figure 4 where the dotted lines indicate the removed edges. Then, we compute the $ub(a_2)$. Since a_2 has three $\mathcal{P}$-neighbors (i.e., a_1, a_3, a_4) and the three $\mathcal{P}$-pair's weights are* 600, 850, *and* 850, *respectively, we have $ub(a_2)$=600, as shown in Figure 4(a) where the red lines indicate the $\mathcal{P}$-pair with the third largest weight among all the $\mathcal{P}$-pairs. Next, we compute the $ub(a_2, S_1)$, by repeating the above process for each $\mathcal{P}$-neighbor of a_2, as depicted in Figure 4(b). As a result,* 600, 850, *and* 850 *are added into set F, implying that $ub(a_2, S_1)$=600. Hence, for a_2, we have $\widehat{f_2}(a_2)$=min$\{ub(a_2), ub(a_2, S_1)\}$=600.*

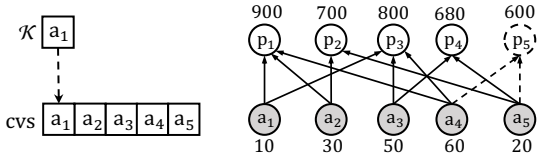
3.2.3 Shrinking $\widehat{f_2}(u)$ to $f_2(u)$. Given a target-keynode u and its $\widehat{f_2}(u)$, we can smoothly shrink $\widehat{f_2}(u)$ to $f_2(u)$, by the following steps. First, we remove all the vertices from V_2 with importance values less than $\widehat{f_2}(u)$. Next, if there is still a $(k, \mathcal{P})$-core containing u in the remaining graph, we set $f_2(u)$=$\widehat{f_2}(u)$. Otherwise, there may exist a skyline influence vector $(\omega(u), f_2(u))$ with $f_2(u) < \widehat{f_2}(u)$. To obtain $f_2(u)$ in this case, we propose a new method, which is different from TypeMax for reducing the computation.

The main idea of this method is to gradually add vertices to $\mathcal{H}[\Phi]$ until there is a $(k, \mathcal{P})$-core containing u in the graph. Specifically, we sort the vertices in V_2, that are removed due to importance values less than $\widehat{f_2}(u)$, in the descending order of importance values. Then, we add vertices back to $\mathcal{H}[\Phi]$ one by one, verify whether the updated sub-HIN forms a $(k, \mathcal{P})$-core, and return the importance value of the last added vertex.

Algorithm 3: Shrink($u, \mathcal{H}[\Phi], \widehat{f_2}(u), V_2$)

```
1  D ← {v2 | (v2 ∈ V2 ∧ ω(v2) < f̂2)};
2  S1 ← remove all vertices in D from H[Φ] and compute (k, P)-core
     in H[Φ];
3  H[Φ'] ← a P-induced sub-HIN of a set of path instances of P
     between vertices in S1 ;
4  if S1 ≠ ∅ and u ∈ S1 then return f̂2(u) ;
5  foreach v ∈ D in reverse order do
6      add v to H[Φ'] ;
7      if there is a (k, P)-core in H[Φ'] then
8          f2(u) ← ω(v) ;
9          break ;
10 return f2(u)
```

Algorithm 3 shows the steps above. We first collect all vertices with importance values less than $\widehat{f_2}(u)$ to a set D, remove all vertices in D from $\mathcal{H}[\Phi]$, and compute the $(k, \mathcal{P})$-core S_1 (lines 1-2). Then, we get a $\mathcal{P}$-induced sub-HIN $\mathcal{H}[\Phi]'$ (line 3). If $S_1 \neq \emptyset$ and it contains u, we have $\widehat{f_2}(u)$=$f_2(u)$ (line 4). Otherwise, we add each vertex v from D back to $\mathcal{H}[\Phi']$ in reverse order one by one. If the updated sub-HIN has a $(k, \mathcal{P})$-core containing u, we record $f_2(u)$=$\omega(v)$ (lines 5-9). Finally, $f_2(u)$ is returned (line 10).



(a) $\mathcal{K}$ and *cvs* (b) $\mathcal{P}$-induced sub-HIN

Figure 5: A running example of Fast2D.

3.2.4 The overall algorithm. To obtain all the skyline influence vectors and HICs, we can process the target-keynodes one by one, where the one with the largest importance value will be processed in the beginning. This process can be further improved by an early-stop strategy. That is, once we obtain a target-keynode corresponding to an HIC that has the largest $f_2(u)$, we can skip verifying the remaining target-keynodes, as stated by Lemma 3.3.

Lemma 3.3. *If there is no vertex $u \in \mathcal{K}$ with $\widehat{f_2}(u)$ larger than the f_2 value of the current skyline influence vector, then all the skyline influence vectors of $\mathcal{H}$ have been obtained.*

Based on the discussions above, we develop a fast algorithm, denoted by Fast2D, in Algorithm 4. Specifically, we first initialize f_2^*=θ_2, which records the f_2 value of the previous skyline influence vector (line 1). Then, we invoke ComputeTK to compute $\mathcal{K}$, *cvs* and $\mathcal{H}[\Phi]$ (line 2). Subsequently, for each target-keynode u, we first

Algorithm 4: Fast2D($\mathcal{H}$, k, $\mathcal{P}$)

input : An HIN $\mathcal{H}$, an integer k, and a meta-path $\mathcal{P}$
output : All HICs and their skyline influence vectors

1 $\mathcal{R} \leftarrow \emptyset$, $f_2^* \leftarrow \theta_2$;
2 $\mathcal{K}, cvs, \mathcal{H}[\Phi] \leftarrow$ ComputeTK($\mathcal{H}$, k, $\mathcal{P}$) ;
3 **foreach** $u \in \mathcal{K}$ *in reverse order* **do**
4 $\quad \widehat{f_2}(u) \leftarrow$ compute the upper bound f_2 of u ;
5 $\quad$ **if** $\max\{\widehat{f_2}(u) | u \in \mathcal{K}\} \leq f_2^*$ **then break**; ;
6 $\quad$ **if** $\widehat{f_2}(u) \leq f_2^*$ **then continue**;
7 $\quad$ **foreach** $v \in cvs$ *starting from* u **do**
8 $\quad\quad$ **if** $v \in \mathcal{K}$ ***and*** $v \neq u$ **then break;** ;
9 $\quad\quad$ add v to $\mathcal{H}[\Phi]$;
10 $\quad f_2(u) \leftarrow$ Shrink($u, \mathcal{H}[\Phi], \widehat{f_2}(u), V_2$);
11 $\quad$ **if** $f_2(u) > f_2^*$ **then**
12 $\quad\quad f_2^* \leftarrow f_2(u)$;
13 $\quad\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{(\omega(u), f_2^*)\}$;
14 **return** $\mathcal{R}$ and the corresponding HICs;

compute $\widehat{f_2}(u)$ (line 4) and add the vertices from u to the next target-keynode in cvs back to $\mathcal{H}[\Phi]$ (lines 7-9). Note that if $\widehat{f_2}(u)$ is not larger than f_2^* and u does not correspond to an HIC, we can skip it (line 6). After that, we use Shrink to obtain $f_2(u)$ and if $f_2(u)$ is larger than f_2^*, we update f_2^*=$f_2(u)$ and append $(\omega(v), f_2^*)$ into $\mathcal{R}$ (lines 10-13). We repeat the above process until all vertices in $\mathcal{K}$ have been traversed. Finally, all the HICs are returned (line 14).

Theorem 3.4. *Algorithm 4 correctly computes all the HICs with 2-dimensional skyline influence vectors.*

Example 5. *Consider Figure 5 with k=3 and $\mathcal{P}$=(APA). Here, $\mathcal{K}$=$\{a_1\}$, cvs=$\{a_1, a_2, a_3, a_4, a_5\}$ and $\mathcal{H}[\Phi]$ only contains $\{p_1, p_2, p_3, p_4, p_5\}$. For the target-keynode a_1, we first add it and its corresponding vertices in the cvs, i.e., $\{a_2, a_3, a_4, a_5\}$, back to the $\mathcal{H}[\Phi]$, where the updated sub-HIN is depicted in Figure 5(b), so we obtain $\widehat{f_2}(a_1)$=680 and delete the paper vertices in V_2 (i.e., p_5) with importance value less than 680. After that, a_5 is also deleted as it only has two $\mathcal{P}$-neighbours, and the remaining vertices will be deleted as well, leading to no (3, $\mathcal{P}$)-core. Thus, we need to add some vertices back to the current sub-HIN. After adding vertex p_5 (with a dashed line cycle) to it, a new (3, $\mathcal{P}$)-core could be formed, as shown in Figure 5(b), and we get $f_2(a_1)$=600.*

Lemma 3.5. Fast2D *completes in $O((n_2+t)\cdot(n_1+m)+n_1\cdot(b_{1,2}+n_2\cdot b_{2,1}))$ time, where t is the number of target-keynodes in $\mathcal{H}$ and the rest of variables have the same meaning as those in Lemma 3.2. Note that t is considerably smaller than n_1 in practice.*

4 ALGORITHMS FOR THE CASE h=3

In this section, we focus on meta-paths with three vertex types (i.e., h=3), so the influence vector is a 3-dimensional vector f=(f_1, f_2, f_3). In the input HIN, we denote the set of vertices with target type by V_1; The sets of vertices with the second and third vertex types are denoted by V_2 and V_3 respectively.

In addition, we use θ_1, θ_2 and θ_3 to denote the smallest importance values in V_1, V_2 and V_3 respectively.

In the following, we present two ICSH algorithms in Sections 4.1 and 4.2. We also briefly discuss how to extend them for solving the cases $h > 3$ in Section 4.3.

4.1 A basic algorithm for the case h=3

To search all HICs for the case h=3, a simple algorithm is to enumerate all the possible importance values of vertices with the second or third type, and by fixing each of them, we invoke an algorithm for processing the case h=2 (e.g., Fast2D in Section 3) to find the 2-dimensional skyline influence vectors for the remaining two vertex types. Afterward, we filter the 3-dimensional influence vectors that are dominated by other influence vectors, which can be easily solved by the traditional skyline algorithm [4], and then get all skyline influence vectors. We denote this algorithm by Basic3D.

Algorithm 5: Basic3D($\mathcal{H}, k, \mathcal{P}$)

input : An HIN $\mathcal{H}$, an integer k, and a meta-path $\mathcal{P}$
output : All HICs and their skyline influence vectors

1 $\mathcal{R} \leftarrow \emptyset$, $\mathcal{T} \leftarrow \emptyset$;
2 $S_1 \leftarrow$ compute the $(k, \mathcal{P})$-core from $\mathcal{H}$;
3 $S_3 \leftarrow$ all the vertices in V_3 that are in the path instances of $\mathcal{P}$ between vertices in S_1;
4 **while** $S_3 \neq \emptyset$ **do**
5 $\quad v \leftarrow \text{argmin}_{v \in S_3}\{\omega(v)\}$;
6 $\quad \mathcal{T} \leftarrow$ Fast2D($\mathcal{H}, k, \mathcal{P}$) ;
7 $\quad$ **foreach** $(f_1, f_2) \in \mathcal{T}$ **do**
8 $\quad\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{(f_1, f_2, \omega(v))\}$;
9 $\quad$ DeleteVertex($\mathcal{H}, S_1, S_3, v$);
10 $\mathcal{R} \leftarrow$ filter all the dominated influence vectors ;
11 **return** $\mathcal{R}$ and the corresponding HICs;

Algorithm 5 presents Basic3D. We first compute the $(k, \mathcal{P})$-core S_1 for $\mathcal{H}$ and collect all vertices in V_3 that are in the path instances of $\mathcal{P}$ between vertices in S_1 (lines 2-3). Then, we iteratively take the vertex u with the smallest importance value in S_3 and compute all the 2-dimensional skyline influence vectors corresponding to $\omega(v)$ by Fast2D, which keeps the 2-dimensional skyline influence vectors in $\mathcal{R}$ (lines 5-8). After that, we remove v from S_3 (line 9). The above process repeats until S_3=$\emptyset$. Finally, we filter all the dominated influence vectors and return $\mathcal{R}$ (lines 10-11).

Remark. In Algorithm 5, we cannot fix the importance values of vertices with target type, and then directly invoke Basic2D or Fast2D to process the other two vertex types, since the minimum $\mathcal{P}$-degree constraint is imposed on the vertices with target type.

Theorem 4.1. *Algorithm 5 correctly computes all the HICs with 3-dimensional skyline influence vectors.*

Lemma 4.2. Basic3D *completes in $O(n_3\cdot(n_2+t)\cdot(n_1+m)+n_1\cdot(b_{1,2}+\sum_{i=2}^{4} n_i\cdot b_{i,i+1}))$ time, where all the variables have the same meanings as those in Lemma 3.5.*

4.2 An advanced algorithm for the case h=3

Basic3D is very intuitive, but it may involve much redundant computation, because it may generate many influence vectors that are dominated by others. To alleviate this issue, we propose a novel algorithm which significantly reduces the number of generated influence vectors, resulting in much higher efficiency. We term this advanced algorithm as Fast3D.

Inspired by Fast2D, our main idea is to first identify all the target-keynodes, and then for each target-keynode u, we derive all the 3-dimensional influence vectors with f_1=$\omega(u)$. To improve the efficiency, we further propose an effective pruning strategy, ensuring that for u, all the derived 3-dimensional influence vectors

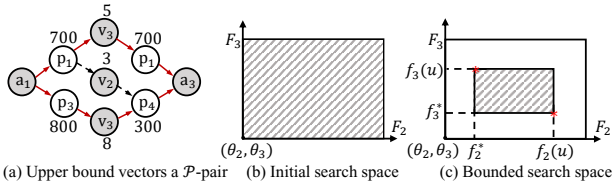
with the first dimension values f_1=$\omega(u)$ are not dominated by each other. As a result, the total number of generated 3-dimensional influence vectors is significantly reduced.

In particular, for each target-keynode v, we can determine the range of its corresponding 2-dimensional influence vector, i.e., $\theta_2 \le f_2 \le F_2$ and $\theta_3 \le f_3 \le F_3$, where F_2 and F_3 are the largest importance values in V_2 and V_3, respectively. We use Figure 6(b) to indicate the entire search space. Obviously, it is inefficient to search in the entire space, so we propose to prune the search space by the upper bound of the influence vectors corresponding to each target-keynode.

• **Upper bound of influence vectors.** According to Definition 10, for the case h=3, there may exist multiple skyline paths between a $\mathcal{P}$-pair, which is different from the case h=2 since it only has one skyline path between the $\mathcal{P}$-pair. Consequently, the weight of each skyline path is a 2-dimensional vector, consisting of the minimum importance value of vertices with the second and third types. A $\mathcal{P}$-pair may have multiple such 2-dimensional vectors, say (x_1, y_1), (x_2, y_2), $\cdots$, (x_r, y_r), so we can introduce a 2-dimensional vector that can serve as an upper bound vector for them:

$$(\widehat{x}, \widehat{y}) = \left(\max_{i\in[1,r]} x_i, \max_{i\in[1,r]} y_i \right). \tag{5}$$

Based on the upper bound vectors of $\mathcal{P}$-pairs, we can compute the upper bound of $f_2(u)$ and $f_3(u)$ of a target-keynode u, i.e., $\widehat{f_2}(u)$ and $\widehat{f_3}(u)$, by the method mentioned in Section 3. Note that $\widehat{x}$ and $\widehat{y}$ are used to compute $\widehat{f_2}(u)$ and $\widehat{f_3}(u)$ respectively.



(a) Upper bound vectors a $\mathcal{P}$-pair (b) Initial search space (c) Bounded search space

Figure 6: Illustrating the main ideas of Fast3D.

Example 6. *Consider the HIN in Figure 6(a) with $\mathcal{P}$=(APVPA). For the $\mathcal{P}$-pair (a_1, a_3), there are three path instances of $\mathcal{P}$ between them, and the skyline paths are $a_1 \to p_1 \to v_3 \to p_1 \to a_3$ and $a_1 \to p_3 \to v_3 \to p_4 \to a_3$ (marked with red line). The weight vectors of these paths are (700, 5) and (300, 8), so $(\widehat{f_2}, \widehat{f_3})$=(700, 8).*

• **Search space pruning strategy**. For each target-keynode u, after deriving $\widehat{f_2}(u)$ and $\widehat{f_3}(u)$, we use Shrink to obtain $f_2(u)$ and $f_3(u)$. Then, we invoke TypeMax to maximize the values of f_3 and f_2 by fixing the values of f_2 and f_3 respectively. As a result, we obtain two 2-dimensional influence vectors, denoted by $(f_2(u), f_3^*)$ and $(f_2^*, f_3(u))$, where $f_2^* \le f_2 \le f_2(u)$, and $f_3^* \le f_3 \le f_3(u)$. Here, the ranges of f_2 and f_3 restrict the overall search space for the remaining 2-dimensional influence vectors, which is much smaller than the entire search space, as illustrated by Figure 6(c). Finally, by following the idea of Basic2D which iteratively maximizes f_2 and f_3, we can find all the remaining influence vectors from these ranges, which are not dominated by each other.

Algorithm 6 presents Fast3D. We first initialize $\mathcal{R}$=$\emptyset$ and $\mathcal{T}$=$\emptyset$ which is used to store all the 2-dimensional skyline influence vectors for each target-keynode (line 1). Then, we invoke ComputeTK to compute $\mathcal{K}$, *cvs* and $\mathcal{H}[\Phi]$ (line 2). Next, we use a for loop to process

Algorithm 6: Fast3D $(\mathcal{H}, k, \mathcal{P})$

input : An HIN $\mathcal{H}$, an integer k, and a meta-path $\mathcal{P}$
output: All HICs and their skyline influence vectors

1 $\mathcal{R} \leftarrow \emptyset$, $\mathcal{T} \leftarrow \emptyset$;
2 $\mathcal{K}$, *cvs*, $\mathcal{H}[\Phi] \leftarrow$ ComputeTK$(\mathcal{H}, k, \mathcal{P})$;
3 **foreach** $u \in \mathcal{K}$ *in reverse order* **do**
4 $\quad \widehat{f_2}(u), \widehat{f_3}(u) \leftarrow$ compute the upper bounds of f_2 and f_3 of u ;
5 $\quad$ **if** $(\widehat{f_2}(u), \widehat{f_3}(u))$ *is dominated* **then** continue;
6 $\quad$ **foreach** $v \in$ *cvs starting from* u **do**
7 $\quad\quad$ **if** $v \in \mathcal{K}$ ***and*** $v \neq u$ **then break**;
8 $\quad\quad$ Add v to $\mathcal{H}[\Phi]$;
9 $\quad \mathcal{T} \leftarrow$ SearchSP$(u, \mathcal{H}[\Phi], \widehat{f_2}(u), \widehat{f_3}(u))$;
10 $\quad$ **foreach** $(f_2, f_3) \in \mathcal{T}$ **do**
11 $\quad\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{(\omega(u), f_2, f_3)\}$;
12 Return $\mathcal{R}$ and the corresponding HICs ;

target-keynodes one by one. Specifically, for each target-keynode u, we compute $\widehat{f_2}(u)$ and $\widehat{f_3}(u)$ (line 4). Then, we check whether $(\widehat{f_2}(u), \widehat{f_3}(u))$ is dominated by the existing 2-dimensional influence vectors (line 5). After that, we add the vertices corresponding to u back to $\mathcal{H}[\Phi]$ and call SearchSP to obtain a set $\mathcal{T}$ which contains all the 2-dimensional skyline influence vectors (lines 6-9). All vectors in $\mathcal{T}$ and f_1=$\omega(u)$ are recorded to $\mathcal{R}$ (lines 10-11). Finally, we obtain all the HICs (line 12).

Algorithm 7: SearchSP$(u, \mathcal{H}[\Phi], \widehat{f_2}(u), \widehat{f_3}(u))$

1 $\mathcal{R}' \leftarrow \emptyset$; ▷ save all 2-dimensional skyline influence vectors of u
2 $(f_2(u), f_3^*), (f_2^*, f_3(u)) \leftarrow$ compute two skyline influence vectors by Shrink and TypeMax based on the $\widehat{f_2}(u)$ and $\widehat{f_3}(u)$;
3 $S_1 \leftarrow$ compute a $(k, \mathcal{P})$-core from $\mathcal{H}[\Phi]$;
4 $S_3 \leftarrow$ all the vertices in V_3 that are in the path instances of $\mathcal{P}$ between vertices in S_1;
5 $T \leftarrow \{v_3 | v_3 \in V_3 \wedge (f_3^* < \omega(v_3) < f_3(u))\}$;
6 **while** $f_2^* \le f_2(u)$ ***and*** $T \neq \emptyset$ **do**
7 $\quad v \leftarrow \text{argmin}_{v\in T}\{\omega(v)\}$;
8 $\quad$ Add v to $\mathcal{H}[\Phi]$;
9 $\quad f_2 \leftarrow$ TypeMax$(\mathcal{H}[\phi], \theta_1, f_2', V_2)$;
10 $\quad$ **if** $(f_2, \omega(v))$ *is not dominated* **then**
11 $\quad\quad \mathcal{R}' \leftarrow \mathcal{R}' \cup \{(f_2, \omega(v))\}$;
12 $\quad\quad f_2' \leftarrow f_2$;
13 $\quad$ Remove v from T;
14 Return $\mathcal{R}'$;

Algorithm 7 shows SearchSP, which computes all the 2-dimension al skyline influence vectors for a target-keynode u. We first initialize $\mathcal{R}'$=$\emptyset$, and use Shrink and Typemax to compute $(f_2(u), f_3^*)$ and $(f_2^*, f_3(u))$ (lines 1-2). Then, we recompute the $(k, \mathcal{P})$-core S_1 of $\mathcal{H}[\Phi]$ and collect a set of vertices S_3 such that each of its vertices is in a path instance of $\mathcal{P}$ between vertices in S_1 (lines 3-4). Next, we append all vertices in S_3 with importance values in $[f_3^*, f_3(u)]$ into T (line 5). Afterwards, we add each vertex v in T back to $\mathcal{H}[\Phi]$ in reverse order one by one, and then compute f_2 of the new updated $\mathcal{P}$-induced sub-HIN by TypeMax (lines 7-9). If $(f_2, \omega(v))$ is not dominated by the other 2-dimensional skyline influence vectors, we append it into $\mathcal{R}'$ and update f_2^*=f_2 (lines 10-12). We remove v from T (line 13) and compute the next 2-dimensional influence

vector by repeating the above process (line 6). Finally, when T=$\emptyset$ or $f_2^* > f_2(u)$, we stop and return $\mathcal{R}'$ (line 14).

Theorem 4.3. *Algorithm 6 correctly computes all the HICs with 3-dimensional skyline influence vectors.*

Lemma 4.4. `Fast3D` *completes in* $O((n_2 + n_3) \cdot (n_1 + m) + n_1 \cdot (b_{1,2} + \sum_{i=2}^{4}(n_i \cdot b_{i,i+1})))$ *time, where all the variables have the same meanings as those in Lemma 3.5.*

4.3 Extending our algorithms for the cases h>3

Our algorithms above can be easily extended for processing the cases h>3 by exploiting the idea of "dimension reduction". Specifically, we first fix one particular vertex type, and then for each importance value of vertices with this type, we compute the skyline influence vectors of the other (h−1) vertex types by the algorithms for the case (h−1). If (h−1) is still larger than 3, we execute the above process recursively; otherwise, `Basic3D` and `Fast3D` can be used directly. Finally, we can obtain the skyline influence vectors by filtering the influence vectors dominated by others and return their corresponding HICs.

5 EXPERIMENTS

We now present the experimental results. Section 5.1 discusses the setup. We discuss the results in Sections 5.2 and 5.3. For lack of space, some additional experimental results for the case h=3 are reported in Appendix B of the technical report [61].

5.1 Setup

Table 3: Datasets used in our experiments.

Dataset	Vertices	Edges	Vertex types	Edge types	Meta-paths
TMDB	7,1978	113,581	7	12	37
DBLP	748,884	1,366,161	4	3	11
IMDB	854,616	3,898,144	4	3	12
DBpedia	5,900,558	17,961,887	413	637	50

Datasets. We use four real HINs: *TMDB*[1] [20], *DBLP*[2] [50], *IMDB*[3] [25], and *DBpedia*[4] [15]. Their statistics, including the numbers of vertices, edges, vertex types, and edge types, are reported in Table 3. *TMDB* is a movie knowledge graph including entities like movies, actors, casts, crews and companies. *IMDB* contains the movie rating records since 2000, and it has four types of vertices (authors, directors, writers and movies). *DBLP* includes publication records in computer science areas, and the vertex types are authors, papers, venues and topics. *DBPedia* contains the data extracted from Wikipedia infoboxes using the mapping-based extraction.

Regarding importance values, for TMDB, we use the worldwide box office grosses of directors as their importance values, and use the popularity (i.e., numbers of Google search results) of movies as their importance values. For other vertices, we use HIVEN model [19] to estimate their importance values, since it is the SOTA model for HIN node importance estimation. For DBLP, we develop a Web Crawler for collecting the h-indexes, citation numbers, h5-indexes, and popularity (i.e., numbers of Google search results) of authors, papers, venues, and topics respectively. For the rest two datasets, the importance values are synthetically generated; that is, for each vertex type, we assign the importance values of all the vertices following the power law distribution, meaning that high degree vertices have higher importance values.

ICSH queries. For the first three datasets, we collect all the possible symmetric meta-paths with lengths less than four. For the remaining one, since it does not have the pre-defined schema, we construct the schema by using its vertex and edge types. Specifically, for each edge, we first collect its end vertex types and edge type, then create two nodes with an edge which are labelled with these vertex types and edge type respectively, and finally obtain a schema network of these new nodes and edges. Notice that each vertex/edge type is allowed to appear only once in the schema. Afterwards, by traversing on the schema network, we collect 50 meta-paths of the highest frequencies (i.e., the number of meta-path instances), where 25 meta-paths are with lengths two and four respectively.

We perform ICSH query for each meta-path where the value of k is set to 5 by default [55]. In the following reported results, each data point is the average result for all meta-paths unless otherwise specified. We implement all the algorithms in Java and run experiments on a machine having an Intel(R) Xeon(R) Gold 6226R 2.90GHz CPU and 256GB of memory, with Ubuntu installed. If an algorithm cannot finish in seven days, we mark its running time as **INF**.

Algorithms. We test the following algorithms.

- **`Basic2D`**: our proposed ICSH algorithm for h=2 (Algorithm 1).
- **`BasicHalf2D`**: this algorithm follows the steps of `Basic2D`, except that `TypeMax` is replaced by `BinaryTypeMax` which finds the maximum value by binary search. The details of `BasicHalf2D` are presented in appendix B of technical report [61].
- **`Fast2D`**: our proposed ICSH algorithm for h=2 (Algorithm 4).
- **`Basic3D`**: our proposed ICSH algorithm for h=3 (Algorithm 5).
- **`Fast3D`**: our proposed ICSH algorithm for h=3 (Algorithm 6).

5.2 Effectiveness evaluation

We analyze the quality of communities from the following aspects:

1. Influence values of communities. We compare the influence of the communities found by our ICSH solution and CSH solution [15], which is the most relevant work and also finds communities using $(k, \mathcal{P})$-core, but does not consider vertex importance. Specifically, we consider two datasets, namely DBLP and TMDB, and for each of them, we first select two meta-paths of two and three vertex types respectively, and then run ICSH queries to get all the HICs. Next, for each HIC, we select the vertex v with the highest importance value as the query vertex to run a CS over HINs (CSH) query, which finds the $(k, \mathcal{P})$-core containing v. Finally, we compute the average importance value of vertices of each type in the communities.

We report the results in Table 4. Clearly, for communities of ICSH queries, their vertices always have higher importance values than vertices of communities of CSH queries, indicating that our ICSH solution is able to find communities with high influence.

2. Compactness, similarity, and density of communities. In this experiment, we aim to compare the compactness, similarity, and density of communities found by the ICSH and CSH solutions [15]. To measure the compactness of communities, a commonly-used metric is the $\mathcal{P}$-distance [15], which is the minimum number

[1] https://www.kaggle.com/tmdb/tmdb-movie-metadata
[2] https://www.aminer.cn/citation
[3] https://www.imdb.com/interfaces/
[4] https://wiki.dbpedia.org/Datasets

Table 4: Vertices' importance values on communities.

Query on DBLP	$\mathcal{P}_1$=(*APA*)		$\mathcal{P}_2$=(*TPVPT*)		
	Author	*Paper*	*Topic*	*Paper*	*Venue*
ICSH	64.57	5,341.2	6,681,984.3	851.7	178.6
CSH	10.98	58.44	2,459,195.8	39.4	50.4
Query on TMDB	$\mathcal{P}_3$=(*MDM*)		$\mathcal{P}_4$=(*GMDMG*)		
	Movie	*Director*	*Genre*	*Movie*	*Director*
ICSH	101.89	5.42	0.35	111.6	6.69
CSH	25.96	0.27	0.29	21.8	0.27

of path instances of $\mathcal{P}$ for linking two vertices (e.g., the $\mathcal{P}$-distance between two vertices linked by an instance of $\mathcal{P}$ is 1). To measure the similarity of community members, we use PathSim [46]. To measure the density of communities, we follow the density measure in [15], which is the number of $\mathcal{P}$-pairs over the number of vertices (here, all the vertices are with the target type) as density.

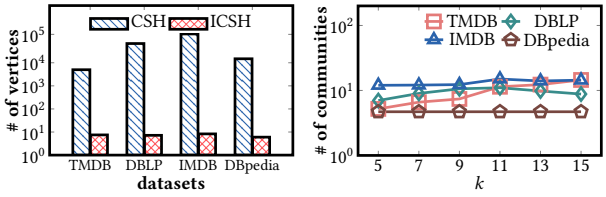
To do the experiments, we set k=5 and for each meta-path, we first run the ICSH query to get all HICs, and then for each HIC, we select the vertex with the highest importance value as the query vertex to run the CSH query. We report the compactness, similarity, and density of communities of ICSH and CSH queries with h=2 on all datasets in Table 5. We omit the results for h=3 since they are similar to those for h=2. We observe that: (1) the communities of ICSH queries have smaller diameters than those of CSH queries, so HICs are more structurally compact and their vertices tend to have closer relationships. (2) the communities of ICSH queries achieve higher similarity values than those of CSH queries. Thus, our ICSH solution can find communities with vertices of higher similarity values. (3) the communities of CSH queries have the lowest densities, while HICs have the highest densities. Therefore, our ICSH solution finds communities with vertices that tend to be more densely connected to each other.

Table 5: Community quality on four datasets for h=2.

Dataset	Diameter		PathSim		Density	
	CSH	*ICSH*	*CSH*	*ICSH*	*CSH*	*ICSH*
TMDB	4.16	1.31	0.04	0.29	92.6	235.8
DBLP	6.80	1.20	0.19	0.34	491.1	869.9
IMDB	14.7	1.28	0.09	0.38	402.0	1,353.3
DBpedia	4.08	1.0	0.50	0.82	2,511.8	2,624.0

3. The sizes and numbers of communities. In this experiment, we first find communities by ICSH and CSH queries (h=2) with settings similar to those in the experiments above, and then report their average sizes in Figure 7(a). Due to the skyline constraint, the average sizes of ICSH communities are around ten, far smaller than those of CSH communities which may have up to 10^5 vertices. Besides, in Figure 7(b), we depict the numbers of searched communities by varying k, which are limited in each dataset, and thus will not make users feel overwhelmed in practice.

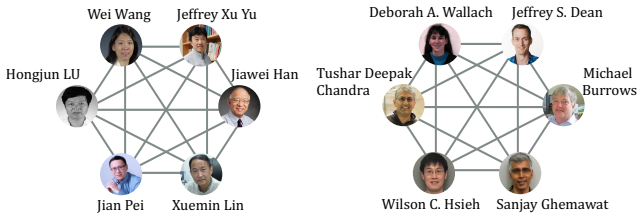
We also evaluate the effect of h and k on the sizes and numbers of communities and show the results in Appendix C of technical report [61]. In line with results of previous works on influential CS [2, 8, 26–29, 31], the community sizes and numbers are small. The main reason is that the number of vertices with very high importance values is often limited. This would be meaningful in practice as existing works [30, 33, 56] have shown that communities with too many vertices may make users feel overwhelmed. Besides, the numbers and sizes of communities increase with h and k. For example, by increasing h=2 to h=3 (k=5), the average community number is changed from 8.5 to 82.1; by increasing k=5 to k=15 (h=2), the average community size is changed from 7.3 to 20.8.



(a) The size of communities (b) The number of communities

Figure 7: The numbers and sizes of communities.

4. A case study. We use a small DBLP network with 411,151 vertices and 719,346 edges, randomly extracted from the original DBLP network. We run an ICSH query and its corresponding CSH query [15] on it by using a meta-path $\mathcal{P}_1$=(*APA*) and setting k=5. Due to the space limitation, we only draw two representative HICs in Figure 8, whose skyline influence vectors are (51, 55) and (12, 6158) respectively. Clearly, the researchers of the first one are more senior than those in the second one, because their h-indexes are higher. Nevertheless, the researchers in the second one have co-authored papers with high importance values. This means that although they may not be as senior as the authors in the first one, they have published works with higher importance values, indicating their teamwork tends to produce higher quality papers.



(a) HIC with influence vector (51, 55) (b) HIC with influence vector (12, 6158)

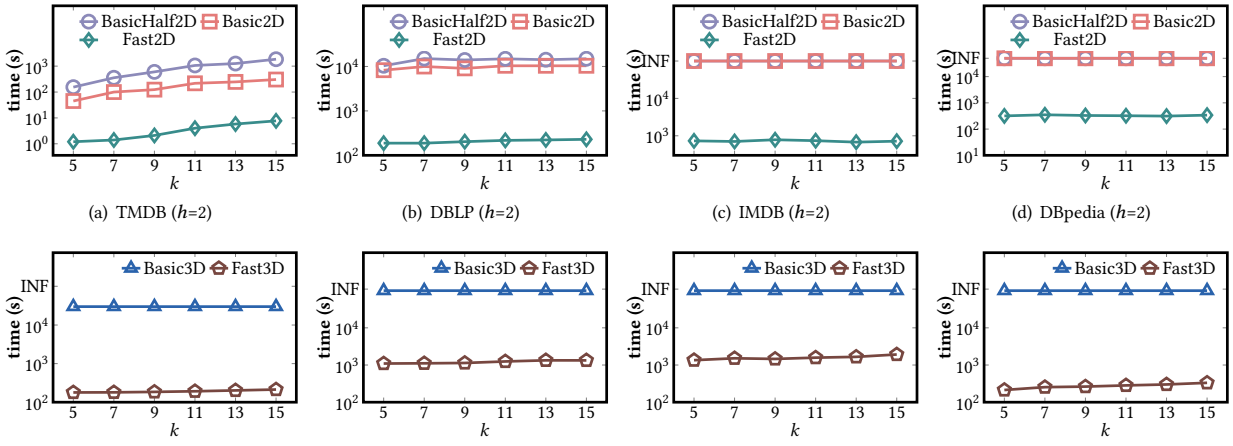
Figure 8: Two HICs on a small DBLP network.

In addition, the average h-index value of authors in the community of CSH query is 12.06, and the average citation number of papers of these authors is 40.64, which are much lower than those of ICSH queries as they do not consider importance values.

5.3 Efficiency evaluation

1. Efficiency of algorithms for the case h=2. Figure 9 depicts the efficiency results by varying k. Clearly, `Fast2D` is up to two orders of magnitude faster than `Basic2D` and `BasicHalf2D`, because they need to iteratively find the influence vector with the largest value in a certain dimension, which involves much redundant computation. In contrast, `Fast2D` can avoid repeated deletion of vertices and reduce the number of deletions to decrease redundant computation.

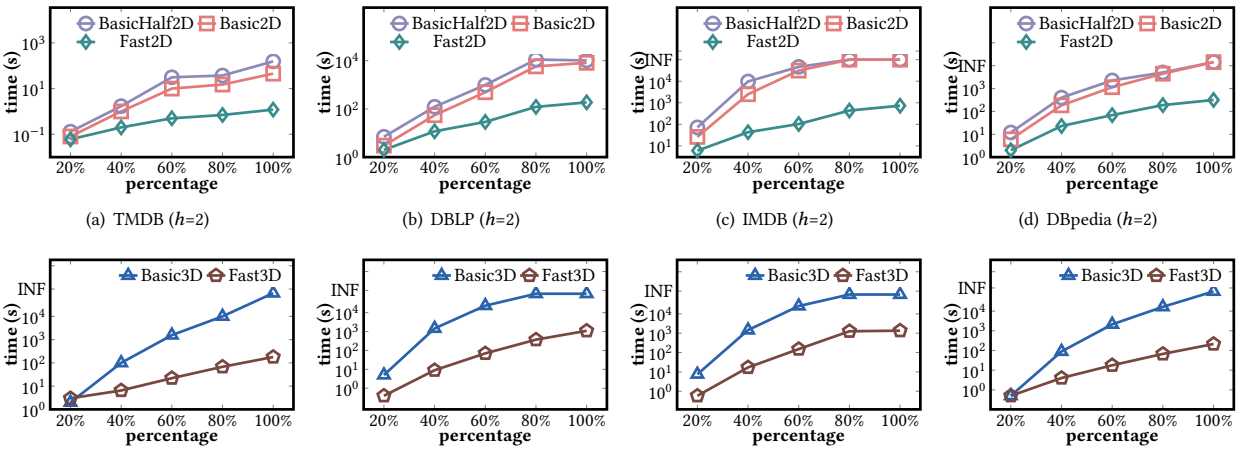
2. Efficiency of algorithms for the case h=3. We show the efficiency results by varying k in Figure 9. We observe that `Fast3D` is at least two orders of magnitude faster than `Basic3D`, since it is not based on dimension reduction, `Basic3D` needs to call `Fast2D` for every possible f_3 value and then filters out all communities with influence vectors dominated by others, which will lead to a huge number of invalid communities to be computed. On the contrary,



(a) TMDB (h=2) (b) DBLP (h=2) (c) IMDB (h=2) (d) DBpedia (h=2)

(e) TMDB (h=3) (f) DBLP (h=3) (g) IMDB (h=3) (h) DBpedia (h=3)

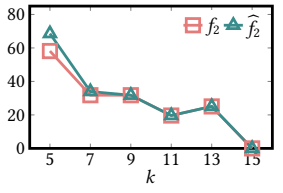
Figure 9: Efficiency results of ICSH algorithms.



(a) TMDB (h=2) (b) DBLP (h=2) (c) IMDB (h=2) (d) DBpedia (h=2)

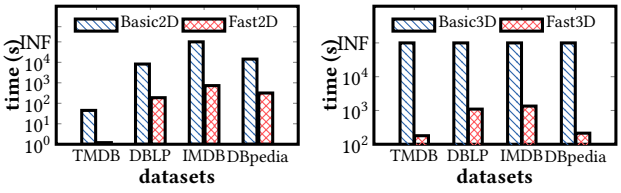
(e) TMDB (h=3) (f) DBLP (h=3) (g) IMDB (h=3) (h) DBpedia (h=3)

Figure 10: Scalability test for ICSH search algorithms.



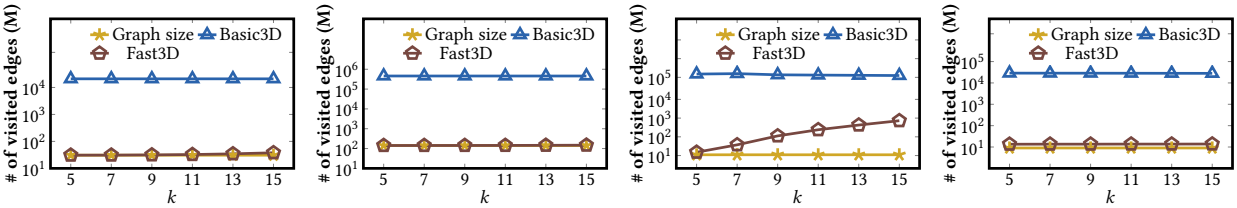
(a) TMDB (b) DBLP

Figure 11: Comparison of upper bound and actual values.



(a) h=2 (b) h=3

Figure 12: Efficiency comparison of ICSH algorithms.



(a) TMDB (h=3) (b) DBLP (h=3) (c) IMDB (h=3) (d) DBpedia (h=3)

Figure 13: The numbers of visited edges of running `Basic3D` and `Fast3D`.

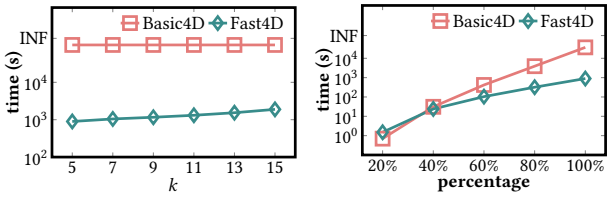
Fast3D only finds all the communities with 2-dimensional skyline influence vectors for each target-keynode, by space search pruning strategy which extremely avoids searching invalid communities.

3. Scalability test. For each HIN, we randomly select 20%, 40%, 60%, 80%, and 100% of its vertices and obtain five sub-HINs induced by these vertices respectively. Then, we run our ICSH algorithms on these sub-HINs, and report the average efficiency results in Figure 10. Generally, their time cost scales linearly with the number of vertices in the graph, indicating good scalability.

4. Analysis of Basic2D and Fast2D by comparing the upper bound and actual values. To further analyze why Fast2D is faster, we run ICSH queries on two datasets, DBLP and TMDB, by using the two meta-paths $\mathcal{P}_1$=(APA) and $\mathcal{P}_2$=(DMD) with k=5, respectively. Figure 11 shows the comparison plot about the actual value of f_2 and the upper bound. We can see that the true value and the upper bound are very close to each other, which is the main reason why Fast2D is faster than Basic2D.

5. Analysis of Basic3D and Fast3D by comparing the size of search space. To measure the size of search space, we run Basic3D and Fast3D on four datasets, during which we count the total numbers of visited edges. Figure 13 shows the results. Clearly, Fast3D is more effective for reducing the search space and avoiding invalid search compared to Basic3D.

6. Dataset sensitivity of our algorithms. We report the time costs of our four algorithms on all datasets in Figure 12. We observe that their time costs are not strictly proportional to the dataset size. For example, DBpedia is the largest dataset, but its time cost is less than that of IMDB. This is because its number of vertex types is much larger than that of IMDB, resulting in fewer vertices for each vertex type which allows the queries to be answered faster.



(a) DBpedia (h=4) (b) DBpedia (h=4)

Figure 14: The results for ICSH search algorithms (h=4).

7. Efficiency and scalability of algorithms for the case h=4. Following the idea in Section 4.3, we design two algorithms Basic4D and Fast4D for the case h=4, based on Basic3D and Fast3D respectively. We collect 10 meta-paths with length six which have 4 vertex types on the DBpedia dataset. We then run the two algorithms and report the efficiency results by varying k in Figure 14(a). Clearly, Fast4D runs much faster than Basic4D. Besides, the scalability results in Figure 14(b) show that Fast4D scales better than Basic4D.

6 RELATED WORK

The two representative groups of community retrieval methods are community detection and community search (CS).

Community detection. The link-based analysis methods [16, 34] are the most representative methods for detecting network communities. However, most of these works focus on homogeneous graphs where vertices are of the same type. Some recent works [41, 42, 45, 47–49, 62] have studied community detection on HINs, which can be roughly divided into two classes depending on the type of community vertex. The first class [6, 42, 45, 49] focuses on detecting clusters, each of which contains objects with multiple types, while the second class [47, 48, 62] aims to generate clusters of objects with a specific type. In [47], Sun et al. proposed an algorithm to generate clusters of a specific type of objects; in [48], a user-guided algorithm is developed to cluster objects of a target type.

Community search (CS). CS aims to query densely connected subgraphs containing a specific vertex or a set of vertices [11, 13, 15, 44, 55]. To measure the structure cohesiveness of a community, people often use the cohesive subgraph models [13], like k-core [1, 3], k-truss [9, 59], k-clique [10, 58] and k-edge connected component [5, 18]. A representative group of CS works is based on the k-core model. For example, in [11, 44], the metric of minimum degree used in k-core is used for CS. Another group of CS works uses the k-truss [9, 59]. For example, in [21, 23], the k-truss-based model is used for CS. Besides, many CS works have considered vertices' attributes (e.g., [7, 13, 22]). Particularly, some works have considered vertices' importance values and studied the problem of influential CS [2, 8, 22, 26–29, 31, 53]. For example, Li et al. studied influential CS on graphs where each vertex has a single importance value [27] and multiple importance values [26]. Some faster algorithms for influential CS have been developed [2, 8].

While CS has been extensively studied, most of the existing works focus on homogeneous networks. Recently, some works have studied CS over HINs [12, 15, 17, 24, 39, 51, 55, 60]. For example, Fang et al. studied CS over HINs by using the (k, $\mathcal{P}$)-core model [15, 25]; Jian et al. [24] searched communities with vertices of multiple types by using relational constraints. Nevertheless, all these works only consider the structural information of HINs, and ignore the importance values of vertices. Thus, it is desirable to study how to effectively perform influential CS over large HINs.

7 CONCLUSIONS

In this paper, we study the problem of influential community search over HINs (or ICSH problem). Conceptually, a highly influential community in the HIN is a set of vertices with the same type, that are not only closely related, but also have high importance values. In particular, we introduce a novel community model for the HIN, called heterogeneous influential community (HIC), which is a set of vertices in a meta-path-based core and its induced sub-HIN has the skyline influence vector. To search HICs, we develop fast algorithms for meta-paths with two and three vertex types, respectively. Our experimental results on four real large HINs show that our solutions are effective and efficient for searching influential communities. In the future, we will study how to maintain HICs efficiently on large dynamic HINs since many real-world HINs are evolving over time. We will consider other aggregate functions (e.g., $sum(\cdot)$ and $avg(\cdot)$) [38] in the community influence vector and develop fast solutions for solving the corresponding problems.

ACKNOWLEDGMENTS

This work was supported in part by NSFC under Grants 62102341, 62202412, and 62272130, Basic and Applied Basic Research Fund in Guangdong Province under Grant 2022A1515010166, Guangdong Talent Program under Grant 2021QN02X826, and Shenzhen Science and Technology Program under Grants JCYJ20220530143602006 and ZDSYS20211021111415025.

REFERENCES

[1] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[2] Fei Bi, Lijun Chang, Xuemin Lin, and Wenjie Zhang. 2018. An Optimal and Progressive Approach to Online Search of Top-K Influential Communities. *Proc. VLDB Endow.* 11, 9 (2018), 1056–1068.

[3] Francesco Bonchi, Arijit Khan, and Lorenzo Severini. 2019. Distance-generalized core decomposition. In *Proceedings of the 2019 International Conference on Management of Data*. 1006–1023.

[4] S. Borzsony, D. Kossmann, and K. Stocker. 2001. The Skyline operator. In *Proceedings 17th International Conference on Data Engineering*. 421–430. https://doi.org/10.1109/ICDE.2001.914855

[5] Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. 2015. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *SIGMOD*. 459–474.

[6] Lu Chen, Yunjun Gao, Yuanliang Zhang, Christian S Jensen, and Bolong Zheng. 2019. Efficient and incremental clustering algorithms on star-schema heterogeneous graphs. In *ICDE*. IEEE, 256–267.

[7] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum co-located community search in large scale social networks. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1233–1246.

[8] Shu Chen, Ran Wei, Diana Popova, and Alex Thomo. 2016. Efficient computation of importance based communities in web-scale networks using a single machine. In *CIKM*. 1553–1562.

[9] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16, 3.1 (2008).

[10] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. 277–288.

[11] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 991–1002.

[12] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-core community search over labeled graphs. *Proceedings of the VLDB Endowment* (2021).

[13] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29, 1 (2020), 353–392.

[14] Yixiang Fang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2021. Cohesive subgraph search over big heterogeneous information networks: Applications, challenges, and solutions. In *SIGMOD*. 2829–2838.

[15] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 6 (2020), 854–867.

[16] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3-5 (2010), 75–174.

[17] Edoardo Galimberti, Francesco Bonchi, and Francesco Gullo. 2017. Core decomposition and densest subgraph in multilayer networks. In *CIKM*. 1807–1816.

[18] Jiafeng Hu, Xiaowei Wu, Reynold Cheng, Siqiang Luo, and Yixiang Fang. 2017. On minimal steiner maximum-connected subgraph queries. *IEEE Transactions on Knowledge and Data Engineering* 29, 11 (2017), 2455–2469.

[19] Chengji Huang, Yixiang Fang, Xuemin Lin, Xin Cao, Wenjie Zhang, and Maria Orlowska. 2022. Estimating Node Importance Values in Heterogeneous Information Networks. In *ICDE*. 846–858.

[20] Han Huang, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. 2021. Representation Learning on Knowledge Graphs for Node Importance Estimation. In *KDD*. 646–655.

[21] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1311–1322.

[22] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[23] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate Closest Community Search in Networks. *Proceedings of the VLDB Endowment* 9, 4 (2015).

[24] Xun Jian, Yue Wang, and Lei Chen. 2020. Effective and efficient relational community detection and search in large dynamic heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1723–1736.

[25] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and ChunshanLi. 2022. Effective Community Search over Large Star-Schema Heterogeneous Information Networks. *Proceedings of the VLDB Endowment* 15, 11 (2022), xxx–xxx.

[26] Rong-Hua Li, Lu Qin, Fanghua Ye, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2018. Skyline Community Search in Multi-valued Networks. In *SIGMOD*. ACM, 457–472.

[27] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential Community Search in Large Networks. *Proc. VLDB Endow.* 8, 5 (2015), 509–520.

[28] Rong-Hua Li, Lu Qin, Fanghua Ye, Guoren Wang, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2020. Finding skyline communities in multi-valued networks. *The VLDB Journal* 29, 6 (2020), 1407–1432.

[29] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2017. Finding influential communities in massive networks. *The VLDB Journal* 26, 6 (2017), 751–776.

[30] Boge Liu, Fan Zhang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2021. Efficient community search with size constraint. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 97–108.

[31] Wensheng Luo, Xu Zhou, Jianye Yang, Peng Peng, Guoqing Xiao, and Yunjun Gao. 2020. Efficient approaches to top-r influential community search. *IEEE Internet of Things Journal* 8, 16 (2020), 12650–12657.

[32] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, and Xiaolin Han. 2022. A Convex-Programming Approach for Efficient Directed Densest Subgraph Discovery. In *SIGMOD*.

[33] Yu-Liang Ma, Ye Yuan, Fei-Da Zhu, Guo-Ren Wang, Jing Xiao, and Jian-Zong Wang. 2019. Who should be invited to my party: A size-constrained k-core problem in social networks. *Journal of Computer Science and Technology* 34 (2019), 170–184.

[34] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.

[35] Webometrics Ranking of World Universities. 2022. Highly Cited Researchers ($h > 100$) according to their Google Scholar Citations public profiles. https://www.webometrics.info/en/hlargerthan100.

[36] Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. 2019. Estimating node importance in knowledge graphs using graph neural networks. In *KDD*. 596–606.

[37] Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. 2020. Multiimport: Inferring node importance in a knowledge graph from multiple input signals. In *KDD*. 503–512.

[38] You Peng, Song Bian, Rui Li, Sibo Wang, and Jeffrey Xu Yu. 2022. Finding Top-r Influential Communities under Aggregation Functions. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1941–1954.

[39] Lianpeng Qiao, Zhiwei Zhang, Ye Yuan, Chen Chen, and Guoren Wang. 2021. Keyword-centric community search over large heterogeneous information networks. In *Database Systems for Advanced Applications: 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11–14, 2021, Proceedings, Part I*. Springer, 158–173.

[40] Chuan Shi, Xiangnan Kong, Philip S Yu, Sihong Xie, and Bin Wu. 2012. Relevance search in heterogeneous networks. In *EDBT*. 180–191.

[41] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2016. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 17–37.

[42] Chuan Shi, Ran Wang, Yitong Li, Philip S Yu, and Bin Wu. 2014. Ranking-based clustering on general heterogeneous information networks by network projection. In *CIKM*. 699–708.

[43] Chuan Shi, Chong Zhou, Xiangnan Kong, Philip S Yu, Gang Liu, and Bai Wang. 2012. Heterecom: a semantic-based recommendation system in heterogeneous networks. In *KDD*. 1552–1555.

[44] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 939–948.

[45] Yizhou Sun, Charu C Aggarwal, and Jiawei Han. 2012. Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. *arXiv preprint arXiv:1201.6563* (2012).

[46] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *Proc. VLDB Endow.* 4, 11 (2011), 992–1003.

[47] Yizhou Sun, Jiawei Han, Peixiang Zhao, Zhijun Yin, Hong Cheng, and Tianyi Wu. 2009. Rankclus: integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*. 565–576.

[48] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S Yu, and Xiao Yu. 2013. Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 7, 3 (2013), 1–23.

[49] Yizhou Sun, Yintao Yu, and Jiawei Han. 2009. Ranking-based clustering of heterogeneous information networks with star network schema. In *KDD*. 797–806.

[50] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD'08*. 990–998.

[51] Ruby W Wang and Y Ye Fred. 2019. Simplifying Weighted Heterogeneous networks by extracting h-Structure via s-Degree. *Scientific reports* 9, 1 (2019), 1–8.

[52] Wikipedia. 2020. Citation impact. https://en.wikipedia.org/wiki/Citation_impact.

[53] Yanping Wu, Jun Zhao, Renjie Sun, Chen Chen, and Xiaoyang Wang. 2021. Efficient personalized influential community search in large networks. *Data Science and Engineering* 6, 3 (2021), 310–322.

[54] Xiaoliang Xu, Jun Liu, Yuxiang Wang, and Xiangyu Ke. 2022. Academic Expert Finding via (K, P)-Core based Embedding over Heterogeneous Graphs. In *ICDE*.

[55] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and efficient truss computation over large heterogeneous information networks. In *ICDE*. IEEE, 901–912.

[56] Kai Yao and Lijun Chang. 2021. Efficient size-bounded community search over large networks. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1441–1453.

[57] Xiao Yu, Xiang Ren, Yizhou Sun, Bradley Sturt, Urvashi Khandelwal, Quanquan Gu, Brandon Norick, and Jiawei Han. 2013. Recommendation in heterogeneous

information networks with implicit user feedback. In *Proceedings of the 7th ACM conference on Recommender systems*. 347–350.

[58] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2017. Index-based densest clique percolation community search in networks. *IEEE Transactions on Knowledge and Data Engineering* 30, 5 (2017), 922–935.

[59] Yikai Zhang and Jeffrey Xu Yu. 2019. Unboundedness and efficiency of truss maintenance in evolving graphs. In *SIGMOD*. 1024–1041.

[60] Alexander Zhou, Yue Wang, and Lei Chen. 2020. Finding large diverse communities on networks: the edge maximum k*-partite clique. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2576–2589.

[61] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2022. Influential Community Search over Large Heterogeneous Information Networks (technical report). https://drive.google.com/file/d/1lUpNkra8mR5natRhtmUEhwlJf732W9-x/view?usp=share_link.

[62] Yang Zhou and Ling Liu. 2013. Social influence based clustering of heterogeneous information networks. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 338–346.