A THE ALGORITHM OF DELETEVERTEX

Algorithm 8: DeleteVertex($\mathcal{H}, S_1, S_2, v$)

```
1  let N(v, S1) be the set of v's P-neighbors in S1;
2  if v ∈ S1 then
3      remove v from S1;
4      foreach u ∈ N(v, S1) do
5          α(u, S1) ← α(u, S1) - 1;
6          if α(u, S1) < k then DeleteVertex(H, S1, S2, u);
7  else
8      S ← a set of P-pairs s.t. v and each P-pair are in a path
         instance of P;
9      remove v from S2;
10     foreach (x, y) ∈ S do
11         if there is no path instance between x and y then
12             α(x, S1) ← α(x, S1) - 1;
13             if α(x, S1) < k then DeleteVertex(H, S1, S2, x);
14             α(y, S1) ← α(y, S1) - 1;
15             if α(y, S1) < k then DeleteVertex(H, S1, S2, y);
```

Algorithm 8 shows the pseudocodes of DeleteVertex, which removes a vertex v and also other vertices with $\mathcal{P}$-degrees less than k after removing v. Specifically, if $v \in S_1$, we decrease the $\mathcal{P}$-degrees of v's $\mathcal{P}$-neighbors, and remove the vertices with $\mathcal{P}$-degrees less than k (lines 2-6). If $v \in S_2$, we first collect all the $\mathcal{P}$-pairs such that v and each $\mathcal{P}$-pair are in a path instance of $\mathcal{P}$ and remove v from S_2 (lines 8-9). For each $\mathcal{P}$-pair (x, y), if there is no path instance linking x and y, we decrease the $\mathcal{P}$-degrees of x and y, and remove the vertices with $\mathcal{P}$-degree less than k (lines 10-15).

B THE ALOGITHM OF BASICHALF2D

Algorithm 9: BasicHalf2D($\mathcal{H}$, k, $\mathcal{P}$)

```
   input : An HIN H, an integer k, and a meta-path P
   output: All HICs and their skyline influence vectors
1  f1 ← θ1, f2 ← ∞, R ← ∅;
2  while f2 > θ2 do
3      f2 ← BinaryTypeMax(H, f1, θ2, V2);
4      f1 ← BinaryTypeMax(H, f1, f2, V1);
5      R ← R ∪ {(f1, f2)};
6      f1 ← min{ω(v1)|v1 ∈ V1 ∧ ω(v1) > f1};
7  return R and the corresponding HICs;

8  Function BinaryTypeMax(H, f1, f2, V):
9      remove vertices {v1|v1 ∈ (V1 ∧ ω(v1) < f1)} and
         {v2|(v2 ∈ V2 ∧ ω(v2) < f2)} from H;
10     S1 ← compute a (k, P)-core from H;
11     S2 ← all the vertices in V2 that are in the path instances of P
         between vertices in S1;
12     f ← min{ω(v)|v ∈ V};
13     while V ∩ (S1 ∪ S2) ≠ ∅ do
14         fmin ← min{ω(v)|v ∈ V};
15         fmax ← max{ω(v)|v ∈ V};
16         while fmin < fmax do
17             f ← (fmin + fmax)/2 ;
18             foreach v in V and ω(v) < f do
19                 DeleteVertex(H, S1, S2, v);
20             if H is ∅ then fmax ← f ;
21             else
22                 fmin ← min{ω(v)|v ∈ V ∧ ω(v) > f};
23     return f;
```

Algorithm 9 illustrates the pseudocode of BasicHalf2D, which iteratively fixes one dimension and maximizes the other dimension using binary search. In this algorithm, the function BinaryTypeMax computes the maximum importance value of a given vertex type. This function employs binary search to identify the largest importance value f, such that an HIC still holds after all vertices with importance values less than f are removed (lines 13-22).

C ADDITIONAL EXPERIMENTS

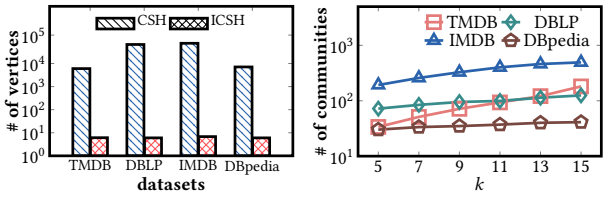
1. Compactness, similarity, and density of communities. We report the compactness, similarity, and density of communities of ICSH and CSH queries with h=3 on all datasets in Table 6.

Table 6: Community quality on four datasets for h=3.

Dataset	Diameter		PathSim		Density	
	CSH	*ICSH*	*CSH*	*ICSH*	*CSH*	*ICSH*
TMDB	2.20	1.03	0.15	0.37	5,980.7	27,322.3
DBLP	3.0	1.0	0.06	0.20	7,906.0	75,182.9
IMDB	9.67	1.18	0.005	0.221	21,611.3	847,562.6
DBpedia	3.83	1.0	0.43	0.72	8,740.4	8,950.3

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.

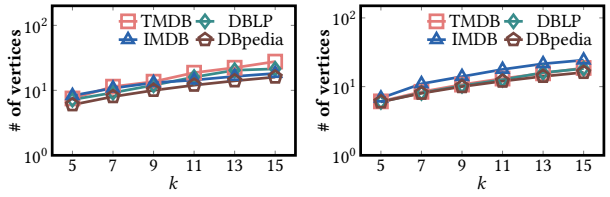
2. The sizes and numbers of communities. In this experiment, we first find communities by ICSH and CSH queries (h=3) with settings similar to those in the experiments above, and then report their average sizes in Figure 16(a). Due to the skyline constraint, the average sizes of ICSH communities are around ten, far smaller than those of CSH communities which may have up to 10^5 vertices. Besides, in Figure 16(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.



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

Figure 15: The numbers and sizes of communities.

In addition, Figures 15(a) and 15(b) demonstrate that the size of searched communities is proportional to k. Specifically, as k increases, the size of communities increases accordingly.



(a) The size of communities (h=2) (b) The size of communities (h=3)

Figure 16: The sizes of communities.

D PROOFS OF LEMMAS AND THEOREMS

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

Proof. We prove this by contradiction. Suppose there is an HIC $\mathcal{G}_0$ with skyline verctor $(f_1(\mathcal{G}_0), f_2(\mathcal{G}_0))$ that cannot be obtained by Algorithm Basic2D. Let the HICs output by Basic2D be $\mathcal{G}_i$ $(1 \le i \le s)$. Without loss of generality, we have $f_2(\mathcal{G}_i) > f_2(\mathcal{G}_{i+1})$ and $f_1(\mathcal{G}_i) < f_1(\mathcal{G}_{i+1})$. $\mathcal{G}_s$ is the last HIC computed by Basic2D, which means that there does not exist an HIC with f_1 value larger than $f_1(\mathcal{G}_s)$ in $\mathcal{H}$. Then we have $f_1(\mathcal{G}_0) < f_1(\mathcal{G}_s)$. It is also note that $f_2(\mathcal{G}_1)$ is the largest f_2 value over all the HICs in $\mathcal{H}$, thus $f_2(\mathcal{G}_0) < f_2(\mathcal{G}_1)$. Since $(f_1(\mathcal{G}_0), f_2(\mathcal{G}_0))$ is a skyline vector, we have $f_1(\mathcal{G}_1) < f_1(\mathcal{G}_0) < f_1(\mathcal{G}_s)$ and $f_2(\mathcal{G}_1) > f_2(\mathcal{G}_0) > f_2(\mathcal{G}_s)$. In other words, there exists an HIC $\mathcal{G}_j$ $(1 \le j \le s-1)$ satisfies that $f_1(\mathcal{G}_j) < f_1(\mathcal{G}_0) < f_1(\mathcal{G}_{j+1})$ and $f_2(\mathcal{G}_j) > f_2(\mathcal{G}_0) > f_2(\mathcal{G}_{j+1})$. It means that $(f_1(\mathcal{G}_0), f_2(\mathcal{G}_0))$ can be obtained by Basic2D, which contradicts the assumption. Therefore, the theorem holds. □

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.

Proof. We first use FastBcore [15] to compute the $(k, \mathcal{P})$-core of $\mathcal{H}$ and build its $\mathcal{P}$-graph, which takes $O(n_1 \cdot b_{1,2} + n_1 \cdot n_2 \cdot b_{2,1})$ time. In each iteration of the while loop, we need to remove all vertices in S_1 and S_2 from $\mathcal{H}$ to maximal the value of f_2 and f_1, which takes $O((n_1+m) \cdot n_2)$ time in the worst case. The number of iterations of the while loop is denoted as s, which is equal to the number of HICs in the input HIN. Thus, the time complexity of Basic2D is $O(s \cdot n_2 \cdot (n_1+m) + n_1 \cdot b_{1,2} + n_1 \cdot n_2 \cdot b_{2,1})$. □

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.

Proof. We prove the lemma by contradiction. Let (f_1^*, f_2^*) be the current skyline influence vector. Assume there exists a new skyline influence vector (f_1', f_2') when there is no $\widehat{f_2}(u)$ in $\mathcal{K}$ larger than f_2^*. Since the target-keynodes are processed in descending order of their importance values, $f_1' < f_1^*$, meaning that f_2' must be larger than f_2^* to satisfy the skyline constraint. Thus, the $\widehat{f_2}$ of f_2' must be larger than f_2^*, which conflicts with the assumption, implying that Lemma 3.3 holds. □

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

Proof. We prove by contradiction. Suppose that there exists an HIC $\mathcal{G}$ with influence vector $(f_1(\mathcal{G}), f_2(\mathcal{G}))$ that cannot be computed by Algorithm 4. There must exist a target-keynode u with $\omega(u)$=$f_1(\mathcal{G})$ by the property of target-keynode. Let the influence vector of the first HIC with f_1 value larger than $\omega(u)$ obtained by Algorithm 4 be (f_1^*, f_2^*). Since $(f_1(\mathcal{G}), f_2(\mathcal{G}))$ cannot be computed by Algorithm 4, we have $f_2(\mathcal{G}) \le f_2^*$ (lines 11-13 in Algorithm 4). Note that $f_1^* > f_1(\mathcal{G})$, so $(f_1(\mathcal{G}), f_2(\mathcal{G}))$ is dominated by the (f_1^*, f_2^*), which conflicts the fact that $(f_1(\mathcal{G}), f_2(\mathcal{G}))$ is a skyline influence vector. Hence, the theorem holds. □

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.

Proof. First, computing the $(k, \mathcal{P})$-core and its corresponding $\mathcal{P}$-graph needs $O(n_1 \cdot (b_{1,2} + n_2 \cdot b_{2,1}))$ time. To obtain all the target-keynodes, we need to iteratively remove all vertices in S_1 which takes $O(n_1+m)$ time in the worst case. Besides, for all target-keynodes, we need $O(t \cdot (n_1+m))$ time to compute $\widehat{f_2}(u)$. In the worst case, we need to invoke Shrink for all target-keynodes which takes $O(n_2 \cdot (n_1+m))$ time. In addition, adding all vertices in cvs to the HIN takes up to (n_1+m) time. Hence, Fast2D takes $O((n_2+t) \cdot (n_1+m) + n_1 \cdot (b_{1,2} + n_2 \cdot b_{2,1}))$ time in total. □

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

Proof. For each possible f_3, we compute its corresponding 2-dimensional skyline influence vector. According to THEOREM 3.4, all the 2-dimensional skyline influence vectors are computed by Fast2D. Hence, all the HICs with 3-dimensional skyline influence vectors are correctly computed by Algorithm 5. □

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.

Proof. First, we need $O(n_1 \cdot (b_{1,2} + \sum_{i=2}^{4} n_i \cdot b_{i,i+1}))$ time to build the $(k, \mathcal{P})$-core's corresponding $\mathcal{P}$-graph. For each vertex in V_3, we need to calculate the 2-dimensional influence vectors by Fast2D, which takes $O(n_3 \cdot (n_2+t) \cdot (n_1+m))$ in the worst case. In addition, removing all vertices in V_3 from $\mathcal{H}$ takes $O(n_3 \cdot (n_1+m))$. Thus, the overall time complexity of Basic3D is $O(n_3 \cdot (n_2+t) \cdot (n_1+m) + n_1 \cdot (b_{1,2} + \sum_{i=2}^{4} n_i \cdot b_{i,i+1}))$. Hence, Lemma 4.2 holds. □

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

Proof. The theorem holds if the corresponding 2-dimensional skyline influence vectors of each target-keynode u can be correctly computed by SearchSP in Algorithm 7. We prove the correctness of SearchSP by contradiction. Suppose that there is a 2-dimensional influence vector (f_2', f_3') that cannot be computed by SearchSP. For each target-keynode u, we compute the corresponding f_2 value for all the possible $f_3 \in [f_3^*, f_3(u)]$ and record all the 2-dimensional skyline influence vectors that cannot be dominated by others. Since (f_2', f_3') cannot be computed by SearchSP, we have $f_3' < f_3^*$ and $f_2' < f_2(u)$ (line 6 in Algorithm 7), which means (f_2', f_3') is dominated by the $(f_2(u), f_3^*)$. It conflicts with the assumption, so the theorem holds. □

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.

Proof. First, computing $(k, \mathcal{P})$-core and its corresponding $\mathcal{P}$-graph needs $O(n_1 \cdot (b_{1,2} + \sum_{i=2}^{4} n_i \cdot b_{i,i+1}))$ time. To obtain all the target-keynodes, we need to remove all vertices in S_1 which takes $O(n_1 + m)$ time in the worst case. For all target-keynodes, we need $O(2 \cdot t \cdot (n_1 + m))$ time to compute $\widehat{f_2}(u)$ and $\widehat{f_3}(u)$. In the worst case, we need to invoke SerachSP for each target-keynode, so this takes $O((n_2 + n_3) \cdot (n_1 + m))$ time. Besides, adding all vertices in *cvs* to the HIN takes up to $(n_1 + m)$ time. Hence, by summarizing the above cost, we obtain Lemma 4.4. □