2023 IEEE 39th International Conference on Data Engineering (ICDE)

Explainable Hyperlink Prediction: A Hypergraph Edit Distance-Based Approach

Hongchao Qin‡, Rong-Hua Li‡, Ye Yuan‡, Guoren Wang‡, Yongheng Dai#
‡Beijing Institute of Technology, China; #Diankeyun Technologies Co., Ltd, China
{hcqin; rhli; yuan-ye; wanggr}@bit.edu.cn; toyhdai@163.com

2023 IEEE 39th International Conference on Data Engineering (ICDE) | 979-8-3503-2227-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDE55515.2023.00386

***Abstract*—Link prediction is a significant technique to generate latent interactions for the applications of recommendation in large graphs. As the interactions to be predicted often occur among more than two objects, we pay attention to solving the novel problem of predicting the interactions in hypergraphs. Previous studies focus mainly on predicting binary relations; most of those techniques cannot be directly applied to predict multiple relations. In this work, we study the problem of edge prediction in hypergraphs, where we use a concept, *Hypergraph Edit Distance* (abbreviated as *HGED*), to measure the similarity of two nodes. Based on *HGED*, we can record a *Hypergraph Edit Path* while searching the optimal edit distance, thus this path enables to explain why one node is similar to another node since their neighborhood structure can be edited to be isomorphic following the edit path. We first propose a general framework which can compute the edit distance of neighborhood structure for two nodes in hypergraph. To improve the efficiency, we propose a *BFS* search-based method with several tightening lower bounds and upper bounds estimation. To predict the multiple relations, we introduce a cluster model in which nodes in each hyperedge are restricted by the hypergraph edit distance. We further present an on-demand algorithm for computing *HGED*, which substantially avoids redundant computations. Finally, we conduct extensive empirical studies on real hypergraph datasets, and the results demonstrate the effectiveness, efficiency and scalability of our algorithms.**

I. Introduction

Link prediction aims at completing the missing links and latent relationships in a graph, which is significant for the applications of recommendation systems in graph-modeled data structure, such as social networks, protein-protein interaction networks and so on. Moreover, each edge in an traditional graph represents a relation of two objects, such that the existing studies on link prediction techniques focus on predicting binary relations in the graph. However, consider the co-operate relations in a research community, where authors usually publish papers in groups of more than two. Therefore, some information would be lost to represent such groups of collaborators by just pairwise edges, and the existing binary relation prediction techniques cannot be directly applied to predict multiple relations. Fortunately, such multiple interactions can be effectively captured by *hyperedges*, an extended notion of edges that join an arbitrary number of entities. Such a novel graph model which consists of nodes and hyperedges is *hypergraph*, which has been widely studied recently. Below, we introduce two cases to illustrate that the real-life multiple relations cannot be easily modeled by a traditional graph, and predicting the missing multiple interactions could be useful in practical applications.

Mining Co-operate Relations in Hypergraph. In a co-operate network, such as DBLP, most researches are done by multiple researchers. Therefore, publications can be regarded as hyperedges, and researchers are nodes in the hypergraph. However, this multiple relation cannot be easily modeled by a traditional graph. This is because that in the traditional graph, the relation of co-operate can be modeled by a clique, and if one author is removed from the clique, other authors will still hold the relations. This contradicts the facts that each publication requires the efforts of all authors, and without one author the publication will not exist. As a consequence, the real-life multiple relations should be modeled by hypergraphs.

Predicting the Expressed Gene in PPI Hypergraph. Genes can acquire mutations in their sequence, leading to different variants, known as alleles, in the population. These alleles encode slightly different versions of a protein, which cause different phenotypical traits. Therefore, genes take effects by groups of protein, and slightly changes of protein will results in the expressed of gene. In a protein-protein interaction hypergraph, proteins and genes can be regarded as nodes and hyperedges. Problems arise that can we predict new genes by the existing recorded genes? Clearly, such a problem can be solved by predicting hyperedges in a protein-protein interaction hypergraph.

As discussed above, modeling and predicting hyperedges could be useful for many practical applications. However, predicting hyperedges are significantly different from predicting edges and it cannot be solved by the existing techniques, since the edges in a traditional graph only include pairwise connection which can be predicted by answering a verification problem whether the similarity of pairwise nodes is larger than a threshold. In addition, to solve the prediction problem, the commonly-used solutions are to embed the nodes and edges into vectors, and then invoke neural networks-based approach to predict missing nodes or edges.

However, those machine learning based models are mostly “black boxes” and can only output the similarity between two nodes. They are not able to explain why two nodes are similar. Therefore, an intuitive and explainable way is to capture the structure features of the nodes and explain how one node differs from the other. One possible solution is to use *Jaccard Similarity* to model the similarity of two nodes, which capture

2375-026X/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDE55515.2023.00386

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

the structural similarity between two nodes. However, this method cannot be used to model the similarity of pairwise node in hypergraph, since the neighborhood structure of one node u in hypergraphs is much more complicated than in traditional graphs. Therefore, to capture how one node differs from the other, we need to characterize the similarity of two sub-hypergraphs by an explainable method first. In this paper, we propose a model which can predict multiple relations in hypergraphs, and also can give an explanation for the reason of getting such results. To the best of our knowledge, we are the first to study the explainable link prediction problem in hypergraphs.

Contributions. In this paper, we formulate and provide efficient solutions to predict edges in a hypergraph by an explainable model. In particular, we make the following main contributions.

***(i)* Novel Model.** First, we propose a novel concept, called *Hypergraph Edit Distance*(HGED), to characterize the similarity of two sub-hypergraphs. Then, the node similarity of nodes u and v is defined by the HGED of the neighborhood structure (ego network) of u and v. Next, we define a new concept called (λ, τ)-hyperedge, which denotes a set of nodes that are similar to each other in terms of HGED with parameters λ and τ. All the *Hypergraph Edit Distances* are accompanied with a *Hypergraph Edit Path*, such that we can explain how can two nodes be similar and a (λ, τ)-hyperedge be formed.

***(ii)* New Algorithms.** To calculate the HGED of two particular hypergraphs, the main technical challenge is how to enumerate and check the final mapping of the nodes and hyperedges. We propose a heuristic *DFS*-based framework to compute HGED first, and then generate a bipartite graph to get an accurate answer. Next, we propose an efficient *BFS*-based method which combined with several non-trivial pruning techniques. Finally, we propose an on-demand verification algorithms to efficiently compute all the (λ, τ)-hyperedges.

***(iii)* Extensive Experiments.** We conduct comprehensive experiments using 6 real-life hypergraphs to evaluate the proposed algorithm under different parameter settings. The results indicate that our algorithms significantly outperform the baselines in terms of the efficiency and accuracy. In addition, we evaluate the efficiency of the proposed algorithms, and the results demonstrate the high efficiency of our algorithms. For example, on a large-scale hypergraph with more than 2M nodes and 4M hyperedges, our algorithm can predict the potential hyperedges in about 6,000 seconds.

Organization. The related work is discussed in Section 2. Section 3 introduces the model and formulation of our problem. The algorithms to efficiently calculating the *Hypergraph Edit Distance* and mining the (λ, τ)-hyperedges are proposed in section 4 and 5. Experimental studies are presented in Section 6, and Section 7 draws the conclusion of this paper.

II. Related Work

In this section, we review recent studies on three related topics: link prediction, hyperlink prediction and graph edit distance.

Link prediction aims to predict whether a pair of nodes in the graph has a link [1]–[3]. Existing methods for link prediction in traditional graphs can be classified into four categories, including the topological-based methods [4], the path-based methods, the supervised methods [5] and the unsupervised methods. *(i) The topological-based methods* mostly predict link based on the topological information of nodes on the links. Some classic measures include *Common Neighbours*, *Cosine Similarity*, *Jaccard Similarity*, *Hub Promoted Index* [6], *Adamic/Adar Index* [7], *Resource Allocation Index* [8], *Leicht-Holme-Newman Index* [9] and so on. *(ii) The path-based methods* can output all paths connecting the considered two nodes. However, the concept of path has different models, such as *Shortest Path*, *Random Walk Path* [10], *Graph Edit Path* and so on. *(iii) The supervised methods* label the nodes and define a loss function to train models that can classify data or predict outcomes accurately [11]–[14]. The prediction model is often in "black box" so the results are hard to explain. *(iv)The unsupervised methods* cluster the nodes in the graph and predict edges by connecting the similar node in the clusters [15]–[19]. Note that, this method is non-explainable since most clustering methods are hard to explain. Our work is similar to the topological-based methods, since we need to capture the neighborhood structure and use the edit distance to represent the similarity. Unlike the above techniques, our work focus on predicting hyperlinks, and we can also predict multiple relations.

Hyperlink rediction aims to predict whether a set of nodes in a hypergraph can be joined by a hyperlink. Recent studies on hyperlink prediction are mainly based on building features or embedding for hypergraphs and invoke neural networks-based approaches for prediction, thus they are all non-explainable. Yoon et al. [20] proposes an incrementally group representing model and use a logistic regression classifier to predict the hypergraphs. Kumar et al. [21] introduce a principle based on the resource allocation process, which is a novel similarity measure in the hypergraphs. Sun et al. [22] put forward a new method to learn high-quality hyperedges using three novel hyperedges distillation strategies automatically and present a novel hypergraph neural networks to predict unobserved links. Zhang et al. [23] propose nonnegative matrix factorization method to infer a subset of candidate hyperedges that are most suitable to fill the training hypernetwork. Jose et al. [24] introduce a novel hypergraphlets-enumeration kernel method on labeled hypergraphs for analysis and learning. All the above mentioned methods are based on machine learning models, in which the predicting process is a "black box" and often hard to explain. Unlike those models, we propose to use hypergraph edit distance to calculate the similarity of two nodes in a hypergraph and develop a (λ, τ)-hyperedge model

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

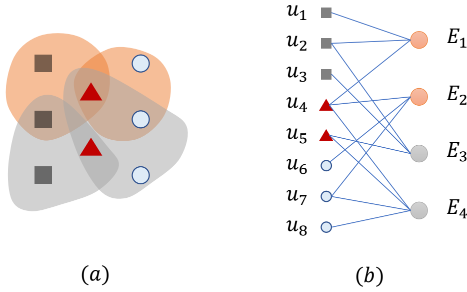


Fig. 1. Example for a hypergraph

for prediction which can capture the neighborhood structure in the hypergraph.

Graph edit distance. Our work is also closely related to the classic *graph edit distance* (which can output a *graph edit path* for explanation). The graph edit distance-based graph similarity search is first studied in [25]. The algorithm to calculate graph edit distance explores the space of all possible mappings between two graphs based on an ordered tree, which is constructed dynamically by iteratively creating successor nodes linked by edges to the currently considered node. The main challenge to calculate graph edit distance is to construct the mappings in the graph as studied in [26]. Besides, some recent works focus on designing index structures such as *q-gram-based index* [27], *subgraph-based index* [28], [29] and so on to filter out as many false-positive graphs as possible. Besides, Chang et al. [30] study the graph edit distance verification problem which verifies whether the graph edit distance between two graphs is no larger than a threshold. In this paper, we propose a significantly *HGED* algorithm, which can capture the edit distance of two hypergraphs.

III. Preliminaries

Let $\mathbb{G} = (V, \mathbb{E}, l)$ be a labeled hypergraph, in which V is the set of nodes, $\mathbb{E}$ is the set of hyperedges $(E_1, E_2...)$ and $l : V \cup \mathbb{E} \to \Sigma$ is a labeling function that assigns each node or hyper edge a label from Σ. Obviously, $|\mathbb{E}| \le 2^{|V|}$. We consider **simple** and **undirected** hypergraph in this manuscript, so one hyperedge $E_x \in \mathbb{E}_{\mathbb{G}}$ is structured as an un-ordered set of nodes $(v_{x_1}, v_{x_2}...)$ and it represents a set of $|E_x|$ nodes that took interaction. For convenience, we record the nodes of each hyperedge $(v_{x_1}, v_{x_2}, v_{x_3}, ...)$ in ascending order such that $(x_1 < x_2 < x_3 < ...)$. Fig. 1 illustrates a hypergraph $\mathbb{G}$ with 8 node and 4 hyperedges.

Here, we introduce some useful concepts for a hypergraph $\mathbb{G}$. Let $V_{\mathbb{G}}$ and $E_{\mathbb{G}}$ be the node set and hyperedge set of $\mathbb{G}$, $n = |V_{\mathbb{G}}|$ and $m = |E_{\mathbb{G}}|$ be the number of nodes and hyperedges, $\mathsf{NEI}_{\mathbb{G}}(v) = v \cup \{u | \exists E_x \in E_{\mathbb{G}}, \{u, v\} \subseteq E_x\}$ be the set of neighbors of node v in $\mathbb{G}$, and $\mathsf{DEG}_{\mathbb{G}}(v) = |\{E_x | v \in E_x, E_x \in E_{\mathbb{G}}\}|$ be the degree of v in $\mathbb{G}$. Considering $E_x \in E_{\mathbb{G}}$, the size of this hyperedge is $|E_x|$ and it also alias the cardinality of E_x. Hypergraph $\mathbb{G}'$ is the sub-hypergraph of $\mathbb{G}$ if $V'_{\mathbb{G}} \subseteq V_{\mathbb{G}}$ and $E'_{\mathbb{G}} \subseteq E_{\mathbb{G}}$. For a given set of nodes $S \subseteq V$, $\mathbb{G}_S$ is referred to as an induced sub-hypergraph of $\mathbb{G}$ from S which satisfies $V_{\mathbb{G}_S} = S$ and $E_{\mathbb{G}_S} = \{E_x | E_x \in E_{\mathbb{G}}, E_x \subseteq S\}$.

Example 1: Fig. 1(a) illustrates a labeled hypergraph $\mathbb{G}$ in which the nodes are label by □, △, ◯, and hyperedges are labeled by different colors (● and ●). A hypergraph can be also represented by a bipartite graph, as shown in Fig. 1(b). We can observe that $\mathsf{NEI}_{\mathbb{G}}(u_4) = \{u_1, u_2, u_4, u_5, u_6, u_7, u_8\}$, $\mathsf{NEI}_{\mathbb{G}}(u_5) = \{u_2, u_3, u_4, u_5, u_7, u_8\}$. □

A. Node Similarity in Hypergraph

In a graph, one edge in a graph represents a binary relation such that the problem of link prediction is to predict the missing edges. This problem can be solved by defining and computing the similarity between two nodes. However, hyperedge represents a multiple relation, thus the definition of node similarity in hypergraph is different and the problem of hyperedge prediction cannot be solved by the existing methods. Below, we introduce some definitions based on one of the most popular graph similarity/distance metrics, Graph Edit Distance (GED), to describe the similarity of nodes in hypergraph. At first, we define the neighbor structure of a given node in a hypergraph.

Definition 1 (Ego Network): Given a hypergraph $\mathbb{G}$ and node $v \in V_{\mathbb{G}}$, the ego network of v, $\mathsf{EGO}_{\mathbb{G}}(v)$, is an induced sub-hypergraph of $\mathbb{G}$ from v's neighbors. Formally, $\mathsf{EGO}_{\mathbb{G}}(v) = \mathbb{G}_{\mathsf{NEI}_{\mathbb{G}}(v)}$.

According to Definition 1, the neighbor structure information of node v can be stored in the ego network of it. As the saying goes, birds of a feather flock together. Intuitively, two nodes are similar if they have similar neighborhoods. Below, we extend an important concept in graph theory, isomorphic, into hypergraphs, such that we can check whether two sub-hypergraphs are the similar.

Definition 2 (Isomorphic in Hypergraph): Given two hypergraphs $\mathbb{G}$ and $\mathbb{G}'$, $\mathbb{G}$ is isomorphic to $\mathbb{G}'$ if there exists a bijective mapping f from $V_{\mathbb{G}}$ to $V_{\mathbb{G}'}$ such that

(i) $\forall v \in V_{\mathbb{G}}$, $l(v) = l(f(v))$;

(ii) $\forall E_x \in E_{\mathbb{G}}$, $l(E_x) = l(f(E_x))$;

(iii) $\forall E_x = (v_{x_1}, v_{x_2}...)$; $E_x \in E_{\mathbb{G}}$ if and only if $(f(v_{x_1}), f(v_{x_2})...) \in E'_{\mathbb{G}}$.

The ego networks of two nodes in $\mathbb{G}$ have low probability to be isomorphic since the condition is very strict. However, a hypergraph can be transformed to be isomorphic to another by a finite sequence of graph edit operations, and then we can define the least-cost edit operation sequence to be GED. However, considering the GED in hypergraphs, the hyperedges contain nodes set of different size, so editing hyperedges with different cardinality will have different cost. Thus, we define the Hypergraph Edit Distance (HGED) as follows.

Definition 3 (Hypergraph Edit Distance): The hypergraph edit distance between two given hypergraphs $\mathbb{G}$ and $\mathbb{G}'$, denoted by $\mathsf{HGED}(\mathbb{G}, \mathbb{G}')$, is the **minimum** sum of edit operations that can transform $\mathbb{G}$ to be isomorphic to $\mathbb{G}'$. In hypergraph, the atomic edit operations include:

(i) inserting/deleting a node or hyperedge (whose cardinality is 0) with a label into/from the graph;

(ii) extending/reducing a hyperedge by adding/deleting one node into/from it;

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

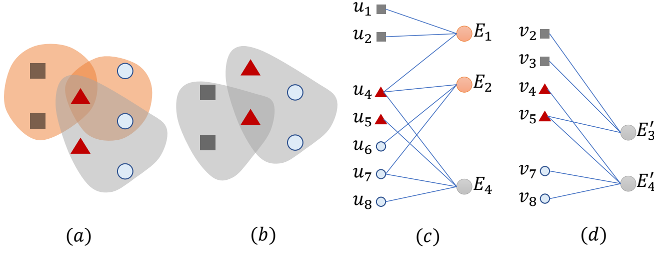


Fig. 2. The HGED of ego networks of u_4 and u_5 in Fig. 1

(iii) changing the label of a node/hyperedge.

Example 2: According to Definition 1, Fig. 2(a), (b) are ego networks of node u_4, u_5 in $\mathbb{G}$ of Fig. 1; Fig. 2(c),(d) are corresponding bipartite graphs of Fig. 2(a), (b). One possible sequence of edit operations for transforming Fig. 2(a) to be isomorphic to Fig. 2(b) is as follows: (1) change the label of E_1 in Fig. 2(c) from ● to ● ('orange' to 'grey'); (2-4) reduce the hyperedge E_2 by deleting nodes u_4, u_6, u_7 in E_2; (5) delete the node u_6 from Fig. 2(a); (6) delete the hyperedge E_2. Thus, the Fig. 2(a) is isomorphic to Fig. 2(b) after 6 edit operations and the mapping of nodes is $\{u_1, u_2, u_5, u_4, u_7, u_8\} \rightarrow \{v_2, v_3, v_4, v_5, v_7, v_8\}$. □

According to Definition 3 and Example 2, we can see that the operation (i) is to add/delete nodes, the operation (ii) is to add/delete edges and the operation (iii) is to change the label of the nodes in the corresponding bipartite graphs. In Example 2, to remove a hyperedge E_2 from $\mathbb{G}$, we need to reduce E_2 by deleting u_4, u_6, u_7 (operation ii), and then delete E_2 since it still hold a label (operation i). Thus, the edit distance of totally deleting E_2 is 4. Besides, in Fig. 2(c), it can be seen that we need to reduce E_2 by deleting node E_2 and edges $(u_4, E_2), (u_6, E_2), (u_7, E_2)$, which also has the counts of 4. In Fig. 2(a), to change the label of E_1, we can change the label of E_1 from orange to grey by one step of operation (iii). Furthermore, we can also represent the process by a deletion and an addition of two operation (i). However, according to Definition 3, we seek for the minimum sum of edit operations, such that we cannot represent a change of label with a deletion and an addition. In conclusion, the three operations in Definition 3 are independent and indispensable.

Problem 1 (Node-Similar Distance). Given a hypergraph $\mathbb{G}$ and node $u, v \in V_{\mathbb{G}}$, the goal of computing the node-similar distance between u and v in $\mathbb{G}$, is to count the hypergraph edit distance between the ego networks of u and v, i.e. $\sigma_{\mathbb{G}}(u, v) = \mathsf{HGED}(\mathsf{EGO}_{\mathbb{G}}(u), \mathsf{EGO}_{\mathbb{G}}(v))$.

Note that, the traditional methods for measuring the similarity of nodes include *Common Neighbours*, *Cosine Similarity*, *Jaccard Similarity*, *Hub Promoted Index* [6], *Adamic/Adar Index* [7], *Resource Allocation Index* [8], *Leicht-Holme-Newman Index* [9] and so on. However, all of those metrics above need to compute the interactions of the neighbors of the corresponding nodes u and v. That is to say, if two nodes have no common neighbors, their similarity is near to zero. In the hypergraph, the neighbors of the nodes directly connected are overlapped such that many of the pair of nodes have almost same neighbors and their similarities are very high; the nodes which are not directly connected have few common neighbors such that their similarities are very low. Nevertheless, the task of hyperedge prediction aims to get some nodes together although they do not have common neighbors, such that the traditional measures of node similarity are not proper to be applied in the task of hyperedge prediction.

Hardness discussions. The key issue to solve problem 1 is to compute the HGED of two given hypergraphs. Unfortunately, computing the exact *GED* in a traditional graph is *NP-hard*. Below, we can show that the traditional *GED* computation is a special case of the HGED problem. Consider a hypergraph in which each hyperedge only links two nodes. Clearly, to compute HGED in this hypergraph is equivalent to computing *GED* in the corresponding bipartite graphs. Thus, the problem of computing HGED is also *NP-hard*.

Although there is a close connection between our problem and the *GED* computation problem, the existing *GED* computation algorithms cannot be directly applied to solve our problem. The reason is that in the hypergraph we cannot get the exact graph edit distance directly by a given mapping of nodes. The details are introduced in section IV-A. □

B. Hyperedge prediction in Hypergraph

In section III-A, we have defined the node similarity in hypergraph by an explainable method. In a graph, we can compute the similarity for a pair of nodes, and then set a threshold to verify whether the edge can be predicted. However, since the hyperedge represents a multiple relationship and it connects more than two nodes, we should design new similarity metrics for connecting the nodes in the predicted hyperedges. Below, we put forward a cluster model which is likely to be a hyperedge.

Definition 4 ((λ, τ)-hyperedge): Given a hypergraph $\mathbb{G}$ and parameters $\tau > 0, \lambda \geq 1$, a node set $S \subseteq V_{\mathbb{G}}$ is a (λ, τ)-hyperedge if and only if $\forall u \in S, \forall v \in \mathsf{NEI}_{\mathbb{G}_S}(u) \Rightarrow \sigma_{\mathbb{G}_S}(u, v) \leq \tau$ and $\forall u, v \in S \Rightarrow \sigma_{\mathbb{G}_S}(u, v) \leq \lambda \times \tau$.

According to Definition 4, a (λ, τ)-hyperedge is a set of nodes in which the node-similar distance between the neighbors is no larger than τ and the node-similar distance between any arbitrary pair of nodes is no larger than $\lambda \times \tau$. The following example illustrates how to derive a (λ, τ)-hyperedge.

Example 3: Consider the hypergraph in Fig. 1 with $\tau = 6, \lambda = 2$. We first start at node u_1, set $S = u_1$, and identify u_1's neighbors $\{u_2, u_4\}$, to check whether the node-similar distance between u_1 and u_2/u_4 is no greater than τ. If the distance of u_1 and u_2/u_4 is no larger than τ, this neighbor will be added into the set S. This process is carried out iteratively until no neighbor of $v \in S$ can be added into S. Then, we need to check whether the node-similar distance of the arbitrary pair of nodes u, v in S is no larger than $\lambda \times \tau$. If the distance of u and v is larger than $\lambda \times \tau$, then u and v will be peeled from S and checked to join in S later. □

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

Problem 2 (Hyperedge prediction). Given a hypergraph $\mathbb{G}$, parameters $\lambda \geq 1, \tau > 0$, the goal of hyperedge prediction is to find all the (λ, τ)-hyperedges in $\mathbb{G}$.

Note that, the parameter τ is set to control the similarity of directly connected nodes in the hyperedges, and parameter λ is set to be lower than the diameter of the graph because it controls the number of hops while searching. The restriction in term of λ is necessary, because it can make (λ, τ)-hyperedges to be overlapping such that we can predict hyperedges of diversity.

Hardness discussions. To predict all the hyperedges in $\mathbb{G}$, we need to compute the node similarity and check whether it can be added into the cluster for each pair of nodes, which is very costly and involves numerous redundant computations of HGED. Therefore, the challenge of problem 2 is how to design a searching framework and pruning strategies which has less redundant computations. □

For convenience, the nodes and edges in hypergraph $\mathbb{G}$ are denoted by $\{u_{i_1}, u_{i_2}, ..., u_{i_n}\}$ and $\{E_1, E_2, ..., E_m\}$; the nodes and edges in another hypergraph $\mathbb{G}'$ are denoted by $\{v_{i_1}, v_{i_2}, ..., v_{i_{n'}}\}$ and $\{E'_1, E'_2, ..., E'_{m'}\}$.

IV. Computing Node Similarity in $\mathbb{G}$

According to problem 1, given a hypergraph $\mathbb{G}$, the node similarity of nodes u and v in $V_{\mathbb{G}}$ is measured by the hypergraph edit distance between $\mathsf{EGO}_{\mathbb{G}}(u)$ and $\mathsf{EGO}_{\mathbb{G}}(v)$, so the key issue of this problem is to compute the HGED of two hypergraphs. In this section, we first introduce a heuristic enumerating framework to compute the HGED of two given hypergraphs. Then, we propose a bipartite graph-based method which can get an accurate answer. Next, we develop a powerful pruning algorithm which can compute the HGED more efficiently by techniques including node re-ranking, upper bound estimations and lower bound estimations.

Algorithm 1: $\mathsf{HGED\text{-}HEU}(\mathbb{G}, \mathbb{G}')$

Input: Two hypergraphs $\mathbb{G}$ and $\mathbb{G}'$
Output: The edit distance of $\mathbb{G}$ and $\mathbb{G}'$ i.e. $\mathsf{HGED}(\mathbb{G}, \mathbb{G}')$

```
1  Let n, n' ← |V_G|, |V_G'|, without loss of generality, n ≥ n';
2  Let V_G = {u_i1, u_i2, ..., u_in}; Extend V_G' to {v_j1, ..., v_jn};
3  I ← {i1, i2, ..., in}; 𝕀 ← ∅;
4  DFS(0, ∅, n, ∅, 𝕀); edc_ ← inf;
5  for I* ← {i*1, i*2, ..., i*n} ∈ 𝕀 do
6      f ← {{u_i*1, u_i*2, ..., u_i*n} → {v_j1, v_j2, ..., v_jn}};
7      edc = EDC-INAC(G, G', f);
8      if edc_ > edc then  edc_ = edc;
9  return edc_ + n − n';
10 Procedure DFS(level, visited, n, f, 𝕀)
11 if level = n then {𝕀 ← 𝕀 ∪ f.keys(); return;}
12 for i ∈ [1 : n] do
13     if !visited[i] then
14         visited[i] ← True; f[level] = i;
15         DFS(level + 1, visited, n, f, 𝕀);
16         visited[i] ← False;
17 Procedure EDC-INAC(G, G', f)
18 /* f = {{u_i1, u_i2, ..., u_in} → {v_j1, v_j2, ..., v_jn}} */
19 edc ← 0;
20 /* node relabeling on G/
21 for k ∈ [1 : n] do
22     if l(u_ik) ≠ l(v_jk) then edc ← edc + 1;
23 /* hyperedge deletion / relabeling on G */
24 for E ← {u_i1, u_i2...u_ih} ∈ E_G do
25     for k ∈ [1 : h] {if f(u_ik) ∉ E_G' then edc ← edc + 1};
26     if l(E) ≠ l({f(u_i1), ..., f(u_ih)}) then edc ← edc + 1;
27 /* hyperedge extending / reducing on G */
28 for E' ← {v_i1, v_i2...v_ih} ∈ E_G' do
29     for k ∈ [1 : h] {if f⁻(v_ik) ∉ E_G then edc ← edc + 1};
30     if l(E') ≠ l({f⁻(v_i1), ..., f⁻(v_ih)}) then edc ← edc + 1;
31 return edc;
```

A. A Heuristic Framework for Computing HGED

To solve the problem of computing the graph edit distance, an intuitive method is to enumerate all the mappings of nodes or edges for the graph G_1 and G_2, then record the mapping which has the fewest edit distances. Since the GED problem is proved to be NP-hard, the SOTA works adopt the filtering-and-verification paradigm to reduce the number of GED verifications, and they mainly focus on designing filtering techniques while using the A^*-GED framework for verification. As the HGED problem is also NP-hard, we have to adopt the filtering-and-verification paradigm which can ensure an exact answer. First, we prove in Lemma 4.1 that there is no node insertion in the optimal sequence (i.e., the sequence with the minimum number) of edit operations that transform $\mathbb{G}$ into $\mathbb{G}'$.

Lemma 4.1: Given hypergraphs $\mathbb{G}$ and $\mathbb{G}'$ with $|V_{\mathbb{G}}| \geq |V_{\mathbb{G}'}|$, there is no node insertion in the optimal sequence of edit operations that transform $\mathbb{G}$ into $\mathbb{G}'$.

Proof: Assume that there is a sequence of edit operations, $O = \{o_1, o_2, ..., o_i, ...\}$, which is the optimal sequence of edit operations that transform $\mathbb{G}$ into $\mathbb{G}'$, and contains a node insertion operation. Without loss of generality, let o_i be a node insertion operation of inserting a node with label x. Since $|V_{\mathbb{G}}| \geq |V_{\mathbb{G}'}|$, there must be a node deletion operation $o_j \in O$. Now, consider another sequence of edit operations O', it differs from O by remove o_i and changing o_j from deleting the node into relabeling the node with label x. It is easy to verify that $|O'| = |O| - 1$, and O' can also transform $\mathbb{G}$ to $\mathbb{G}'$, which contradicts that O is optimal. □

According to lemma 4.1, we can extend $V_{\mathbb{G}'}$ by *null* node to have the same size as $V_{\mathbb{G}}$ in the algorithm at first. Below, we introduce a heuristic enumeration framework and abbreviate it as HGED-HEU.

Algorithm 1 first extends $V_{\mathbb{G}'}$ by *null* node to have the same size as $V_{\mathbb{G}}$ based on lemma 4.1 (lines 1-2). Then, it initializes an array I to store the index of $V_{\mathbb{G}'}$ and $\mathbb{I}$ to store the permutations of I (line 3). Next, the procedure DFS generates all the permutations of the array I (line 4), which is detailed in lines 10-16. For each permutation I^* in $\mathbb{I}$, we construct a mapping f of nodes in line 6. Next, we calculate the edit distance of f and record the lowest value in all loops (lines 7-9). Finally, the algorithm returns $\underline{edc} + n - n'$, which is the

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

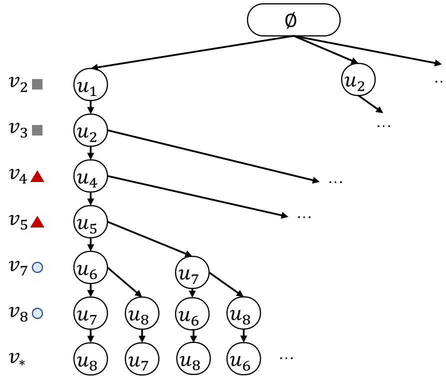


Fig. 3. The *DFS* search tree of mapping $\text{EGO}_{\mathbb{G}}(v_4)$ to $\text{EGO}_{\mathbb{G}}(v_5)$ in Fig 2.

lowest edit distance plus the edit cost of extending $V_{\mathbb{G}'}$. In lines 10-16, the procedure DFS build a search tree by *Depth First Search*. The root node of the search tree is at level 0, and represents an empty mapping $\emptyset$. The procedure records the *level*, the *visited* node set, the mapping size n, the candidate mapping f and the result mappings $\mathbb{I}$. An intuitive example of the *DFS* search is shown in Fig. 3. In lines 17-31, the procedure EDC-INAC computes the edit cost of the node mapping $f = \{\{u_{i_1}, u_{i_2}, ..., u_{i_n}\} \to \{v_{j_1}, v_{j_2}, ..., v_{j_n}\}\}$ from $V_{\mathbb{G}}$ to $V_{\mathbb{G}'}$. If the label of node u_{i_1} is not equal to the label of v_{j_1}, the recorded value edc increases by 1 (lines 21-22). Consider the hyperedge $E \in E_{\mathbb{G}}$, if the mapping edge is not existed, then the deletion of each node will result in an increase of edit distance (line 25), and if the label of the mapping edge is not same to the label of E, edc will add up by 1 (line 26). Similar, consider the hyperedge $E' \in E_{\mathbb{G}'}$, edit distance increase if the corresponding edges are not in $E_{\mathbb{G}}$ (lines 27-30). However, we hold the following observation by analyzing the edited hyperedges.

Example 4: As shown in Fig. 3, since the number of nodes in $\text{EGO}_{\mathbb{G}}(v_5)$ is 6 but the number of nodes in $\text{EGO}_{\mathbb{G}}(v_4)$ is 7, we first extend $\text{EGO}_{\mathbb{G}}(v_5)$ by a node v_*. Next, we traverse the nodes $\{u_1, u_2, u_4, u_5, u_6, u_7, u_8\}$ by depth first and we can get all the permutations of them. □

Observation 4.1: Procedure EDC-INAC can give an instance of edit distance but it is not minimum for the mapping f.

The observation 1 holds because in procedure EDC-INAC, we simply extend or reduce one whole hyperedges once they are not mapped, but we can slightly change one hyperedges by adding/deleting one node into/from it. In the following section, we propose a method which can compute the accurate edit cost between the two hypergraphs.

Complexity of HGED-HEU. For a hypergraph $\mathbb{G}$ with n nodes and m hyperedges, the time complexity of HGED-HEU (Algorithm. 1) is $O(m \times n!)$.

Proof: First, Algorithm. 1 needs $O(n)$ to extend the nodes (line 1). For each index of nodes (line 5), it needs to invoke the procedure EDC-INAC to compute the edit cost (line 7). While computing the edit cost, we need $O(n+m)$ to record the node/edge relabeling operations (lines 20-23). and $O(m)$ to record the hyperedge insertion/deletion since the average cardinality of hyperedges is no larger than 10

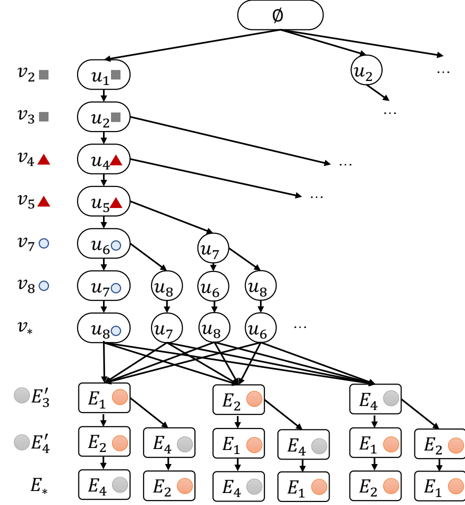


Fig. 4. The accurate edit cost computation of $\text{EGO}_{\mathbb{G}}(v_4)$ to $\text{EGO}_{\mathbb{G}}(v_5)$

(lines 23-30). As the number of node mapping is the full permutations, so the algorithm needs to invoke EDC-INAC for $O(n!)$ times. In conclude, HGED-HEU needs the time complexity of HGED-HEU. □

B. A Bipartite Graph-Based Computation for HGED

To compute the accurate edit cost from $\mathbb{G}$ to $\mathbb{G}'$ under the node mapping f, we need to consider the mapping of all the hyperedges. Below, we introduce the algorithm 2, which is abbreviated as EDC.

Algorithm 2 first generates a bipartite graph $B_{\mathbb{G}}$ construct by $\mathbb{G}$, in which the nodes of $B_{\mathbb{G}}$ consist of $V_{\mathbb{G}} \cup \{u_{E_1}, ...u_{E_m}\}$ and the edges consist of $\{(u_i, u_{E_j}) | u_i \in E_j\}$. Similarly, let $B_{\mathbb{G}'}$ be a bipartite graph construct by $\mathbb{G}'$. Then, the algorithm initializes the mapping of hyperedges by I_E (line 3), and generate the permutations of I_E into $\mathbb{I}_E$ by procedure DFS (line 4). For each mapping of hyperedges I_E^* (line 5), we combine the node mapping f and I_E^* into a fixed mapping f' (line 6). If the label of node or edge is not mapped (lines 7-11), the maintained edit cost edc increases 1. For each edge (u_i, u_{E_j}) in the constructed bipartite graph $B_{\mathbb{G}}$, if edge $(f'(u_i), f'(u_{E_j}))$ is not in another bipartite graph $B_{\mathbb{G}'}$, we must delete an edge from $B_{\mathbb{G}}$ and edc will add up by 1 (lines 12-14). Furthermore, $B_{\mathbb{G}}$ do not have the edge in $B_{\mathbb{G}'}$, we must add an edge into $B_{\mathbb{G}}$ and edc will increase 1 (lines 15-17). Finally, $\underline{edc}$ record the minimum edit cost and the algorithm returns $\underline{edc}$ (line 19).

Example 5: Recall Fig. 2, we first add a node v_* and a hyperedge E_* into the mapping list, which can make sure that the length of nodes and hyperedges of $\mathbb{G}$ and $\mathbb{G}'$ are the same. Next, the algorithm HGED-DFS enumerates the nodes by depth first. Unlike Fig. 3, the accurate edit cost computation also enumerates the permutations of all the hyperedges, such that we can count the node/edge relabeling and edge deletion/insertion to compute the accurate edit cost. For example, the total cost of node relabeling of mapping $f = \{v_2, v_3, u_4, u_5, v_7, v_8, v_*\} \to \{u_1, u_2, u_4, u_5, u_6, u_7, u_8\}$ is 1. And the total costs of edge relabeling by mapping

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2: EDC($\mathbb{G}, \mathbb{G}', f$)

Input: Two hypergraphs $\mathbb{G}$, $\mathbb{G}'$ and a mapping f from $V_{\mathbb{G}}$ to $V_{\mathbb{G}'}$ ($f = \{\{u_{i_1}, ..., u_{i_n}\} \rightarrow \{v_{j_1}, ..., v_{j_n}\}\}$)
Output: the minimum edit cost from $\mathbb{G}$ to $\mathbb{G}'$ under f

```
1  Let B_G be a bipartite graph construct by G, in which
     V_BG ← V_G ∪ {u_E1, ...u_Em}; E_BG ← {(u_i, u_Ej)|u_i ∈ E_j};
2  Similarity, B_G' is a bipartite graph construct by G';
3  Let I_E ← {E_1, E_2, ...E_m}; 𝕀_E ← ∅;
4  DFS(0, ∅, m, 𝕀_E); edc_ ← 0;
5  for I*_E ∈ 𝕀_E do
6      f' ← {{u_i1, u_i2..., u_in, u*_E1, u*_E2, ..., u*_Em} →
         {v_j1, v_j2..., v_jn, v_E'1, v_E'2, ..., v_E'm}}; edc ← 0;
7      /* node/edge relabeling on B_G */
8      for k ∈ [1 : n] do
9          if l(u_ik) ≠ l(v_jk) then edc ← edc + 1;
10     for h ∈ [1 : m] do
11         if l(u*_Eh) ≠ l(v_E'h) then edc ← edc + 1;
12     /* edge deletion on B_G */
13     for each (u_i, u_Ej) ⊂ E_BG do
14         if (f'(u_i), f'(u_Ej)) ∉ E_BG' then edc ← edc + 1;
15     /* edge insertion on B_G */
16     for each (v_i, v_Ej) ⊂ E_BG' do
17         if (f'^-(v_i), f'^-(v_Ej)) ∉ E_BG then edc ← edc + 1;
18     if edc_ > edc then edc_ = edc;
19 return edc_;
```

Algorithm 3: HGED-BFS($\mathbb{G}, \mathbb{G}'$)

Input: Two hypergraphs $\mathbb{G}$ and $\mathbb{G}'$
Output: The edit distance of $\mathbb{G}$ and $\mathbb{G}'$ i.e. HGED($\mathbb{G}, \mathbb{G}'$)

```
1  Let n ← max(|V_G|, |V_G'|), m ← max(|E_G|, |E_G'|);
2  Let V_BG be {u_i1, u_i2, ..., u_in, u_E1, u_E2, ..., u_Em}; and V_BG'
     be {v_i1, v_i2, ..., v_in, v_E'1, v_E'2, ..., v_E'm};
3  Let I ← ReRank{i_1, i_2, ..., i_n, E_1, ..., E_m};        [Strategy 1]
4  edc‾ ← estimate a upper bound for edit cost;            [Strategy 2]
5  Q ← {[0, ∅, ∅, I, 0]}; R ← ∅;
6  while Q ≠ ∅ do
7      [level, visited, f, I, edc] ← Q.pop();
8      for x ← [level : n + m] and !visited[x] do
9          C ← cost of editing from u_I[x] to v_x;
10         edc_ ← estimate lower bound of mapping the
             remaining nodes;                                [Strategy 3]
11         if edc + C + edc_ ≤ edc‾ then
12             if x + 1 = n + m then R = R ∪ (edc + C(x));
13             else Q.push([level + 1, visited ∪ u_I[x], f ∪
                 {f[level] ← I[x]}, I, edc + C(x));
14 return min(R)
```

$\{E_3', E_4', E_*\}$ into $\{E_1, E_2, E_4\}$ or $\{E_1, E_4, E_2\}$ are 3 or 2. However, the costs of edge deletion/insertion by mapping $\{E_3', E_4', E_*\}$ into $\{E_1, E_2, E_4\}$ under node mapping f is $2 + 1 + 4 = 7$, including $\{v_2, v_3, v_5\} \rightarrow \{u_1, u_2, u_4\}$, $\{v_4, v_5, v_7, v_8\} \rightarrow \{u_4, u_6, u_7\}$ and $\emptyset \rightarrow \{u_4, u_5, u_7, u_8\}$. And the cost of edge deletion/insertion by mapping $\{E_3', E_4', E_*\}$ into $\{E_1, E_4, E_2\}$ under node mapping f is $2 + 2 + 3 = 7$, including $\{v_2, v_3, v_5\} \rightarrow \{u_1, u_2, u_4\}$, $\{v_4, v_5, v_7, v_8\} \rightarrow \{u_4, u_5, u_7, u_8\}$ and $\emptyset \rightarrow \{u_4, u_6, u_7\}$. So the total edit cost of $\{f, \{E_3', E_4', E_*\} \rightarrow \{E_1, E_2, E_4\}\}$ is $1 + 3 + 7 = 11$, and the total edit cost of $\{f, \{E_3', E_4', E_*\} \rightarrow \{E_1, E_4, E_2\}\}$ is $1 + 2 + 7 = 10$. □

As Fig. 4 and Example. 4 show, EDC($\mathbb{G}, \mathbb{G}', f$) can compute the accurate edit cost from $\mathbb{G}$ to $\mathbb{G}'$ under the node mapping f. In addition, we modify the algorithm 1 by replacing the EDC-INAC into EDC, and abbreviate the new algorithm as HGED-DFS.

Complexity of HGED-DFS. For a hypergraph $\mathbb{G}$ with n nodes and m hyperedges, the time complexity of HGED-DFS is $O(m \times n! \times m!)$.

Proof: The HGED-DFS is modified by algorithm 1 with replacing the EDC-INAC into EDC. So it also need to invoke procedure EDC for $O(n!)$ times. However, in EDC (Algorithm 2), we also need to enumerate the permutations of all hyperedges, such that the time complexity of EDC is $O(m \times m!)$. So the total time complexity of HGED-DFS is $O(m \times n! \times m!)$. □

C. A BFS Pruning Enumeration for Getting HGED

Although we can use HGED-DFS to compute the HGED of two hypergraphs, it still has two limitations. *(i)* it has to enumerate all the permutations of nodes and edges to get all candidate mappings. *(ii)* it is hard to find some lower bounds while using the *DFS* metric to search the mappings. To overcome those two limitations, we propose an improved algorithm called HGED-BFS. The striking features of HGED-BFS are twofold. On one hand, it needs not to enumerate all the permutations of nodes and edges. Instead, it calculates edit cost during the enumeration and early terminates while the current edit cost is larger than the currently optimal answer. On the other hand, HGED-BFS searches the mappings by *Breadth First*, such that we can compute several lower bounds for different levels of the search tree, which will significantly speed up the enumerations. Below, the detailed description of HGED-BFS is shown in Algorithm 3.

Algorithm 3 first extends $V_{\mathbb{G}}$ and $V_{\mathbb{G}'}$ to contain all the indexes of nodes and hyperedges in $\mathbb{G}$ and $\mathbb{G}'$ (lines 1-2). Then, it samples several mappings to get an estimation upper bound for the edit cost between $\mathbb{G}$ and $\mathbb{G}'$ (line 4). Next, the algorithm initializes a queue Q, to search the index of $V_{\mathbb{G}}$ by *Breadth First* (line 5), and perform several powerful pruning strategies during the enumeration (line 10). There are three strategies to speed up the enumeration. And an intuitive example of the *BFS* search is shown in Fig. 5.

Example 6: Unlike Fig. 4, we first re-rank the nodes and hyperedges by their degrees and cardinalities (**[Strategy 1]**), such that the $V_{B_{\mathbb{G}}}$ is $\{u_4, u_5, u_1, u_2, u_7, u_6, u_8, E_4, E_1, E_2\}$. Then, we can estimate the upper bound by the original matching order (**[Strategy 2]**) such that $\overline{edc} = 1 + 2 + 4 + 5 + 3 = 15$. Next, we traverse the $V_{B_{\mathbb{G}}}$ by breadth first. It enumerates the first level and record the cost of editing from the current u_x to the mapping node v_x, such that

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

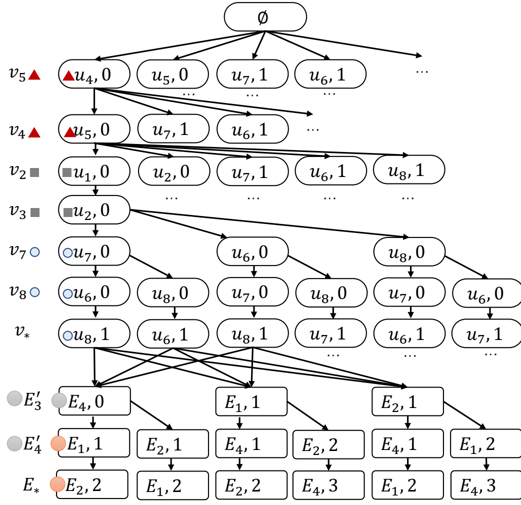


Fig. 5. Examples for computing HGED-BFS($\mathsf{EGO}_{\mathbb{G}}(v_4), \mathsf{EGO}_{\mathbb{G}}(v_5)$)

$(u_4, 0)(l(u_4 = l(v_5)), (u_5, 0)(l(u_5 = l(v_5)), (u_7, 1)(l(u_7 \neq l(v_5))$ and so on. However, it also computes the label based and hyperedge based lower bounds $\underline{edc}_f^{lb} + \underline{edc}_f^{heb}$ (**[Strategy 3]**). If the sum of the accumulated edc, the current editing cost C and the lower bounds of the remaining mapping $\underline{edc}$ is no larger than the upper bound $\overline{edc}$, then the algorithm pushes the array into the queue Q which makes sure they will be searched in the *BFS* tree. □

[Strategy 1] *Rerank the nodes*. The matching order can be optimized at the beginning of the initialization (line 3 of algorithm 3). It is mainly based on four intuitions: *(i)* the node with the higher degree should be mapped first; *(ii)* the nodes or hyperedges with the same label should be put together because they are more likely to be be similar nodes or hyperedges; *(iii)* the nodes should be mapped before the hyperedges; and *(iv)* the hyperedge with the higher cardinality should be be mapped first. So, the first node should be the one with the highest degree and the following nodes should have the same labels with the first one. The optimized matching order has a great influence to the running time in practice, which will be shown in Section. 6 below.

[Strategy 2] *Upper bound estimation*. The upper bound of the HGED can be estimated by two methods. The first one is that we can set a threshold τ because we are not interested in two sub-hypergraphs which are not much similar to each other. In our experiments we find that the parameter setting of τ will largely reduce the running time of the whole algorithms. The second one is that we can sample some random matching orders and compute the edit cost of the given matching. We can consider the four intuitions in the strategy 1 above, and we also find that in some situations the simply ranked matching order has the similar HGED as the optimal mappings.

[Strategy 3] *Lower bound estimation*. The lower bound that has been extensively used in the existing algorithms [25], [30]. However, since the mappings in our proposed HGED model contain nodes and hyperedges, we propose the following lower bounds.

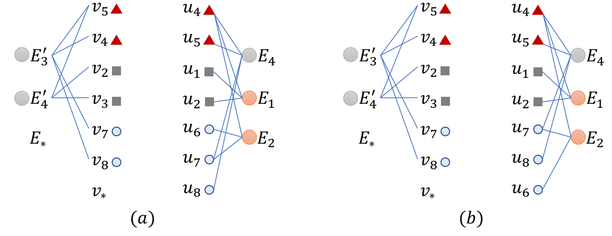


Fig. 6. Computing the edge deletion/insertion of two mappings

Definition 5 (label based lower bound): Given two labeled hypergraphs $\mathbb{G}$ and $\mathbb{G}'$, the label based lower bound is $\underline{edc}_f^{lb} = \Psi(l(V_{\mathbb{G}}), l(V_{\mathbb{G}'})) + \Psi(l(E_{\mathbb{G}}), l(E_{\mathbb{G}'}))$ where $\Psi(S_1, S_2) = \max\{|S_1|, |S_2|\} - |S_1 \cap S_2|$.

According to Definition 5, we can have that if the label set of nodes and hyperedges in $\mathbb{G}$ are $\{A, A, B, C\}$ and $\{a, a, b\}$, and the label set of nodes and hyperedges in $\mathbb{G}'$ are $\{A, B, B, C\}$ and $\{b, b, c\}$, the label based lower bound is $4-3+3-1=3$.

Definition 6 (hyperedge based lower bound): Given two labeled hypergraphs $\mathbb{G}$ and $\mathbb{G}'$, the hyperedge based lower bound is $\underline{edc}_f^{heb} = \max \Sigma_{i=1}^{\max(|E_{\mathbb{G}}|, |E'_{\mathbb{G}}|)} ||E_i| - |E'_i||$ where $E_i \in E_{\mathbb{G}}$ and $E'_i \in E_{\mathbb{G}'}$.

According to Definition 6, for example, if the hyperedges $\{E_1, E_2, E_3, E_4\}$ in $\mathbb{G}$ have the cardinalities of $\{4, 2, 5, 3\}$; and the hyperedges $\{E'_1, E'_2, E'_3, E'_4\}$ in $\mathbb{G}'$ have the cardinalities of $\{6, 4, 4, 3\}$. Then the hyperedge based lower bound is $6-5+4-4+4-3+3-2=3$.

We can find that the label based lower bound and the hyperedge based lower bound have no overlaps, such that the $\underline{edc} = \underline{edc}_f^{lb} + \underline{edc}_f^{heb}$. We show an example of searching with estimating the lower bound in Fig.6.

Example 7: Consider the hypergraph edit distance of $\mathsf{EGO}_{\mathbb{G}}(v_4)$ and $\mathsf{EGO}_{\mathbb{G}}(v_5)$). Recall Fig.5 , we get two mappings in Fig.5 and show them as Fig.6(a) and Fig.6(b). Fig.6(a) correspondences to the mapping f' from $\{v_5, v_4, u_2, u_3, v_7, v_8, v_*, E'_3, E'_4, E_*\}$ to $\{u_4, u_5, u_1, u_2, u_6, u_7, u_8, E_4, E_1, E_2\}$. Therefore, the edit cost of node label for f' is 1 (which can be acquired in the *3rd* item of level 7 in Fig.5), the edit cost of hyperedge label for f' is 2 (which can be acquired in the first item of level 10 in Fig.5), and the edge insertion/deletion cost of f' is $2+0+3=5$, so the edit cost of f' is $1+2+5=8$. Fig.6(b) correspondences to the mapping f'' from $\{v_5, v_4, u_2, u_3, v_7, v_8, v_*, E'_3, E'_4, E_*\}$ to $\{u_4, u_5, u_1, u_2, u_7, u_8, u_6, E_4, E_1, E_2\}$. So, the edit cost of node label for f' is 1 (which can be acquired in the *2nd* item of level 7 in Fig.5), the edit cost of hyperedge label for f' is 2, and the edge insertion/deletion cost of f' is $0+0+3=3$, so the edit cost of f'' is $1+2+3=6$. In addition, we can find that the edge insertion/deletion cost of f' equals the hyperedge based lower bound, and the edit cost of node/hyperedge labels also equals the label based lower bound. Therefore, the f'' in Fig.6(b) is no doubt the optimal hypergraph edit distance. We can also observe that the value 6 is same as the count of edit operations in Example 2, so

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

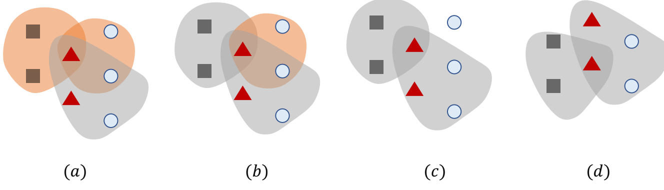


Fig. 7. Explainable Hypergraph Edit Path

those edit operations in Example 2 is the optimal *hypergraph edit path*. □

Complexity of HGED-BFS. For a hypergraph $\mathbb{G}$ with n nodes and m hyperedges, the time complexity of HGED-BFS is $O(m \times 2^{h_v} \times 2^{h_e})$, in which h_v and h_e are maximum search levels of node and hyperedges.

Proof: Algorithm 3 needs to maintain a queue Q (line 5), as can be seen in Fig. 5, the length of Q is no larger than $O(\times 2^{h_v} \times 2^{h_e})$, in which h_v and h_e are maximum search levels of node and hyperedges. And calculating the edit cost for each node or hyperedge needs $O(m)$. So, the total time complexity of HGED-BFS is $O(m \times 2^{h_v} \times 2^{h_e})$. □

Note that, according to the pruning rule of strategy 2 and 3, h_v and h_e are near to the upper bound and lower bound of the edit operations, such that they are usually not larger than 10. We will show the running time in practice at Section 6.

D. An Explainable Hypergraph Edit Path

Example 6 introduces that the edit operations in Example 2 is the optimal *hypergraph edit path*. Here, we introduce how we can explain the node similarity by a given *hypergraph edit path*. Fig. 7 shows the hypergraph edit path from the ego network of node u_4 into the ego network of u_5 in Fig. 2. We can explain that the transformation from Fig. 7(a) to Fig. 7(d) as follows. First, one group changes their interests from • to • ('orange' to 'grey'), as shown in Fig. 7(b). Then, the remaining people which are interested in • disappear, but there is only one person left (Fig. 7(c)). Next, the last person also leaves and there only have two groups of •. So, this transformation from (a) to (d) may be a rise of emerging topics in a hypergraph, in which the old topics (•) and the person who persists the old topic are disappear from the network, but the remaining people fall in interest with a new topic. And the total distance of those two relations is 6, as calculated in Example 2 and 6.

For each HGED, we can always output a *Hypergraph Edit Path* and we can try to explain it as the descriptions above.

V. Hyperedge Prediction

In this section, we develop an efficient algorithm to compute all the (λ, τ)-hyperedges. The basic idea of our algorithm is as follows. According to the definition of (λ, τ)-hyperedges, we can find that a (λ, τ)-hyperedge denotes a set of nodes in which the similar distance of the directly connected nodes is no larger than τ and the similar distance of each pair of nodes is no larger than $\lambda \times \tau$. An intuitive method is to compute all the node-similar distance for each pair of nodes in the hypergraph to get a connected component S, then enumerate all the nodes and check which node can be added into S. Before introducing the algorithm, we present a lemma which helps to transform from the connected component S into a (λ, τ)-hyperedge.

Algorithm 4: HEP$(\mathbb{G}, \tau, \lambda)$

Input: Hypergraphs $\mathbb{G}$ and parameters τ, λ
Output: The predicted (λ, τ)-hyperedge

```
1  𝕊 ← ∅;
2  for u ∈ V_𝔾 s.t. u ∉ 𝕊 do
3      S ← {u}; Q ← {u};
4      while Q ≠ ∅ do
5          v ← Q.pop();
6          for w ∈ NEI_{𝔾_S}(v) and w ∉ S do
7              if HGED(EGO_{𝔾_S}(w), EGO_{𝔾_S}(v)) ≤ τ then
8                  S.add(w); Q.push(w);
9      𝕊.add(S);
10 for each S ∈ 𝕊 do
11     Search S by BFS as lines 5-13 of Algorithm 3; or by
       DFS as lines 10-16 of Algorithm 1;
12     If |PATH_S(u,v)| ≥ λ, then we need to check whether
       HGED(EGO_{𝔾_S}(u), EGO_{𝔾_S}(v)) ≤ λ × τ;
13     Return possible set S' ⊆ S in which each u, v ∈ S'
       satisfies HGED(EGO_{𝔾_{S'}}(u), EGO_{𝔾_{S'}}(v)) ≤ λ × τ.
```

Lemma 5.1: Given a hypergraph $\mathbb{G}$, nodes $u, v \in V_{\mathbb{G}}$ and parameters $\tau > 0, \lambda \geq 1$, if there is a path $\{u'_0, u'_1, ..., u'_p\}$ ($u'_0 = u$, $u'_p = v$, $p < |\lambda|$) satisfying $\sigma_{\mathbb{G}}(u'_i, u'_{i+1}) \leq \tau$ for $i \in [0 : |\lambda| - 1]$, then u, v are in at least one (λ, τ)-hyperedge.

According to lemma 5.1, we can first compute several connected components $\mathbb{S}$ from one node v by search all the neighours, and then filter and enumerate the nodes whose are over λ-hops from v. The detail of the HEP algorithm is shown as follows. For each node in the hypergraph $\mathbb{G}$, we initialize a set $\mathbb{S}$ to record the candidate connected component S, a queue Q to store the nodes in the enumeration path (line 2). Then, the algorithm checks all the neighbor of the nodes in Q and attempts to find all the nodes which directly connects to S and the similar distance of the connection is no larger than τ (lines 3-8). After all the nodes are considered, the connected component S is added into $\mathbb{S}$ (line 9) and the algorithm restarts to find another connected component (line 2).

Next, the algorithm searches the candidate $\mathbb{S}$ by *BFS*, similar as lines 5-13 of Algorithm 3; or by *DFS*, similar as lines 10-16 of Algorithm 1. However, in each enumeration, it first checks whether v is over λ-hops from u (line 12). If so, it checks whether the node u to be added into the enumerated set S', satisfies that $\forall v \in S'$, $\mathrm{HGED}(\mathrm{EGO}_{\mathbb{G}_{S'}}(u), \mathrm{EGO}_{\mathbb{G}_{S'}}(v)) \leq \lambda \times \tau$. If $\mathrm{HGED}(\mathrm{EGO}_{\mathbb{G}_{S'}}(u), \mathrm{EGO}_{\mathbb{G}_{S'}}(v)) \leq \lambda \times \tau$, the node u will be added into S'. Otherwise, the enumeration continues but u is not added into S'.

Complexity of HEP. For a hypergraph $\mathbb{G}$ with n nodes and m hyperedges, the time complexity of HEP is $O(n^2 \times m \times 2^{h_v} \times 2^{h_e})$, in which h_v and h_e are maximum search levels of node and hyperedges.

Proof: Algorithm 4 needs to compute the HGED for at most $O(n^2)$ times, since we can store the HGED results for each

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

TABLE I
STATISTICS OF DATASETS

Dataset	$\lvert V\rvert = n$	$\lvert\mathbb{E}\rvert = m$	$\overline{\lvert E\rvert}$	$\lvert\hat{E}\rvert$	$\lvert l(V)\rvert$
PS	242	12,704	2.4	2	11
HS	327	7,818	2.3	2	9
MO	73,851	5,446	24.2	5	1,456
WM	88,860	69,906	6.6	5	11
TVG	172,738	233,202	4.1	3	160
AMZ	2,268,231	4,285,363	17.1	8	29

pair of nodes. However, as HGED needs $O(m \times 2^{h_v} \times 2^{h_e})$ to compute the node-similar distance for each pair of nodes, the time complexity of HEP is $O(n^2 \times m \times 2^{h_v} \times 2^{h_e})$. □

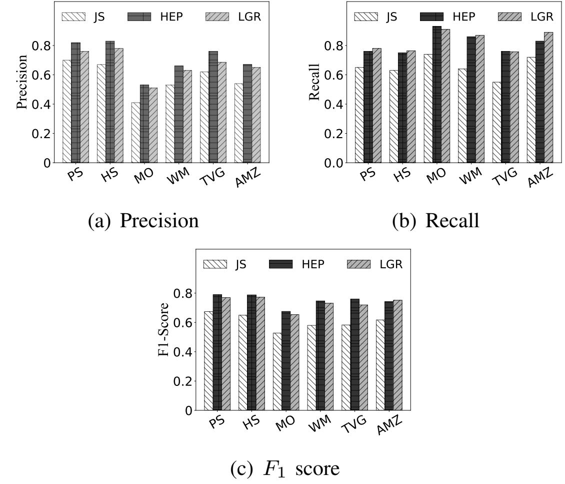
VI. EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the proposed algorithms. We implement seven different algorithms for comparison:

- HGED-HEU is a baseline that computes an instance of edit distance of two sub-hypergraphs, which is proposed in Section 4.1.
- HGED-DFS is an algorithm which can compute the edit distance of two sub-hypergraphs by *DFS* search, which is proposed in Section 4.2.
- HGED-BFS is an algorithm which can compute the edit distance of two sub-hypergraphs by *BFS* with several pruning strategies, which is proposed in Section 4.3.
- JS is a baseline in which we use *Jaccard Similarity* to compute the similarity for each pair of nodes, and then use the HEP framework to predict hyperedges.
- HEP-BFS is the HEP in which we use a *BFS* search to compute the HGED, which is proposed in Section 5.
- HEP-DFS is the HEP in which we use a *DFS* search to compute the HGED, which is proposed in Section 5.
- LGR [20] is a *SOTA* logistic regression classifier to predict the hyperedges with *L2* regularization, which uses the n-order expansion of a hypergraph to capture the features.

All algorithms are implemented in Python. All the experiments are conducted on a Linux kernel 4.4 server with an Intel Core(TM) i5-8400@3.80GHz processor and 32 GB memory. When quantity measures are evaluated, the test was repeated over 5 times and the average is reported here.

Datasets. We use 6 different real-world hypergraphs in the experiments. The detailed statistics of our datasets are summarized in Table I, where $\overline{|E|}$ denotes the mean of the hyperedge size, $|\hat{E}|$ denotes the median of the hyperedge size, and $|l(V)|$ denotes the number of node/hyperedge classes. All the datasets are downloaded from https://www.cs.cornell.edu/~arb/data/. PS *(primary school)* and HS *(high school)* are datasets of the contact in primary and high school in which each hyperedge corresponds to a group of people that were all in proximity of one another at a given time, and each node is labeled as a teacher or the classroom to which the student belongs. MO *(mathoverflow)* are sets of questions answered by users on *Math Overflow*, where labels are question tags. WM *(walmart)* are sets of products bought on Walmart shopping trips, where labels are departments of products. TVG *(trivago)* are sets of hotels clicked on in a Web browsing session, where labels are the countries of the accomodation. AMZ *(amazon)* are sets of products reviewed by users on Amazon, where labels are product categories.



(a) Precision (b) Recall

(c) F_1 score

Fig. 8. Effectiveness results of different algorithms

Goodness metrics. We use the method of *Data Validation Testing* to test the effectiveness of the hyperedge prediction. We split the hyperedges into a testing dateset and a validation dateset by a ratio of 3 to 1, and conduct our proposed method to predict the hyperedges on testing dateset. Then, we record the predicted results of testing dateset and compare them with the validation dateset. Next, we use a *Precision-Recall* metric to evaluate the prediction quality.

Precision P is defined as the number of true positives T_P over the number of true positives plus the number of false positives F_P, such that $P = \frac{T_P}{T_P + F_P}$.

Recall R is defined as the number of true positives T_P over the number of true positives plus the number of false negatives F_N, such that $R = \frac{T_P}{T_P + F_N}$.

These quantities are also related to the F_1 score, which is defined as the harmonic mean of precision and recall, such that $F_1 = 2 \times \frac{P \times R}{P + R}$.

A. Effectiveness Evaluation

Exp-1. Effectiveness results of different algorithms. Fig. 8 shows the results of the proposed hyperedge prediction algorithm HEP with $\lambda = 3, \tau = 5$ (as HEP-BFS/HEP-DFS are the variants of the algorithm HEP (Algorithm 4) in which we use HGED-DFS/HGED-BFS to calculate the node-similar distance. Both of them output the same results, thus we use HEP to denote our method in effectiveness testings). The baseline JS also has parameter λ and τ, and we set $\lambda = 3$, $\tau = 0.8$ such that the ratio between the intersection and the union of the neighbor nodes is no less than 0.8. In LGR [20], we set $n = 3$ and extract 6 features. Similar results can also be observed using the other parameter settings. We can see from Fig. 8(a-c) that JS performs poorly in all situations, because JS can only capture limited structural information of the nodes while modeling the similarity. As shown in Fig. 8(a), HEP outperforms LGR in terms of the *Precision* metric in

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

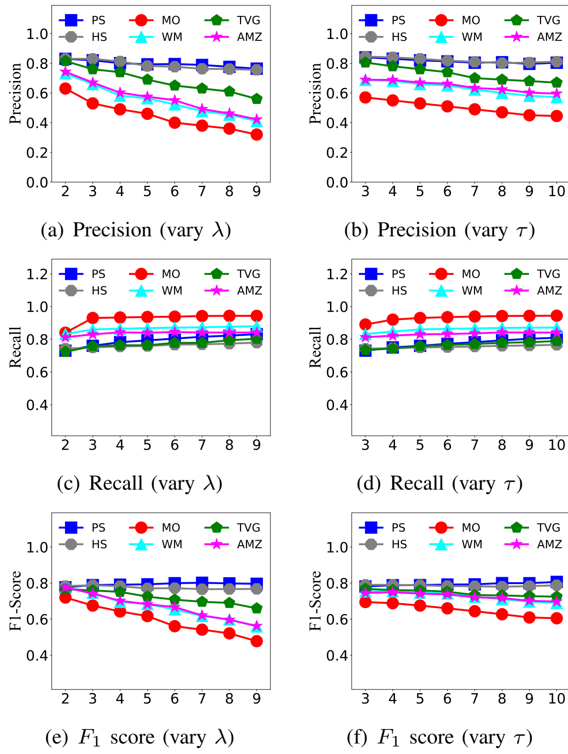
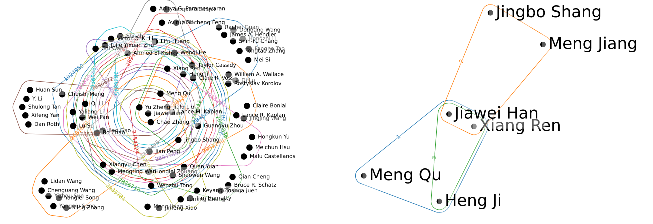


Fig. 9. Effectiveness of HEP with varying parameters on all datasets



(a) The hyperedges which contain Prof. Jiawei Han in year 2016 (b) The prediced hyperedges

Fig. 10. Case study on DBLP

all datasets. This is because our proposed algorithm HEP uses the structural information to predict hyperedges, which is more accurate than the feature-based methods. Note that, in the datasets PS and HS, the precision value is near to 0.8, but in dataset MO, the precision value is much lower. The reason is that the hyperedges in PS and HS are much denser than that in MO; thus, the prediction is more accurate in a hypergraph with higher density. However, in Fig. 8(b), the *Recall* of the hyperedge prediction by LGR is slightly higher than HEP. This is because that LGR considers the cases where each candidate hyperedge has cardinality 3, 4, ... 10. However, the cardinalities of most hyperedges predicted by HEP are 3 and 4. Therefore, LGR tends to find more hyperedges than HEP which results in better performance in terms of the recall metric. In Fig. 8(b), consider the dataset AMZ, where LGR outperforms HEP. This is because the labels on AMZ is generated by a machine learning method, which uses similar features as LGR. Therefore, we believe that this is the characteristics of this dataset that allow LGR to be better.

In Fig. 8(c), we can observe that the *F1-Score* of HEP is better than LGR in most datasets. The results indicate that our model has better comprehensive performance.

Exp-2. Effectiveness results of HEP with varying parameters. Here we study how the parameters affect the effective performance of our algorithm. Fig. 9 shows the results of HEP with varying parameters. We vary λ from 2 to 9 with a default value of 3 in the testing, and τ from 3 to 10 with a default value of 5. Unless otherwise specified, the values of the other parameters are set to their default values when varying a parameter. As can be seen in Fig. 9(a)-(b), the *Precision* increases with growing λ and τ. This is because when λ or τ values increase, the T_p value in *Precision* will be larger. However, we can find the change of the *Precision* with varying τ is not very significant, compared to the parameter λ. The reason may be that λ has a direct impact on the size of the predicted hyperedges, and the predicted hyperedges change more frequently when varying λ. In Fig. 9(c)-(d), the *Recall* value decreases with growing λ and τ. This is because when λ and τ increase, more possible hyperedges are found and the incorrectly-rejected instances F_N in *Precision* will be lower. In Fig. 9(e)-(f), we can find that the *F1-Score* decreases with growing λ and τ, and the change is more slight while varying τ. It indicates that the performance of our model is better when λ and τ are lower.

Exp-3. Case study on DBLP. Here, we conduct a case study to show that the proposed method can indeed predict possible co-author relations on DBLP. Fig. 10(a) shows 24 hyperedges which contain Prof. Jiawei Han (http://hanj.cs.illinois.edu/) in year 2016. In Fig. 10(a), each node represents a researcher and each hyperedge is a publication on DBLP in year 2016. Fig. 10(b) shows parts of the $(3, 5)$-hyperedges which is predicted by our proposed model. In Fig. 10(b), we can find that the hyperedge of the 'orange' color contains researchers "Jiawe Han", "Xiang Ren", "Jingbo Shang" and "Meng Jiang". Interestingly, we find that in year 2017, the above authors have co-authored a KDD paper [31] and an arXiv paper [32], but they did not collaborate together in year 2016. Besides, we can also record all the edit path for each pair of nodes in the hyperedges, which makes the calculation of similarity explainable. In conclusion, the proposed (λ, τ)-hyperedge model can indeed correctly predict the missing hyperedges in hypergraphs.

B. Efficiency Evaluation

Exp-4. Running time of HGED computation methods. Table. II evaluates the running time of HGED-HEU, HGED-DFS and HGED-BFS. We invoke each algorithm for 1,000 pairs of nodes in different datasets to compute the HGED, and then record the average time of computing once. From Table. II, we can see that HGED-BFS is much faster than HGED-HEU

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

TABLE II
RUNNING TIME (S) OF DIFFERENT HGED COMPUTATION ALGORITHMS

Dataset	HGED-HEU	HGED-DFS	HGED-BFS
PS	0.23	0.43	**0.23**
HS	0.25	0.48	**0.14**
MO	69.3	80.23	**10.3**
WM	78.3	140.5	**30.3**
TVG	407.1	623.2	**130.3**
AMZ	7,308.1	8,234.3	**623.8**

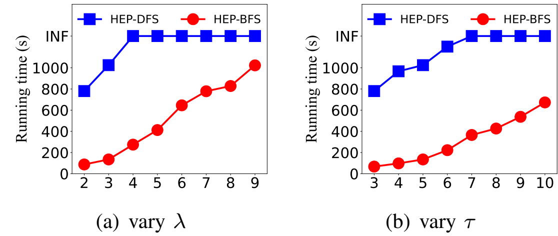
TABLE III
RUNNING TIME (S) OF HEP-DFS V.S. HEP-BFS

	HEP-DFS	HEP-BFS	LGR
PS	123.2	**15.3**	116.9
HS	100.3	**13.2**	99.5
MO	1025.2	**134.7**	1132.5
WM	3690.2	**492.4**	2332.5
TVG	9823.1	**1539.3**	5521.3
AMZ	> 1 day	**6687.5**	> 1 day

and HGED-DFS. This is because that we use three strategies to speed up finding the minimum HGED. Note that, since the neighbors in a hypergraph usually has a similar neighborhood, we can set the upper bound HGED to be 10 in most situations. HGED-HEU is slightly faster than HGED-DFS, since it not need to enumerate the permutations of hyperedges. As can be seen, on AMZ, HGED-DFS takes only 8,234.3 seconds and our best algorithm HGED-BFS only consumes 623.8 seconds. These results confirm that our proposed algorithms are indeed very efficient on large real-life hypergraphs.

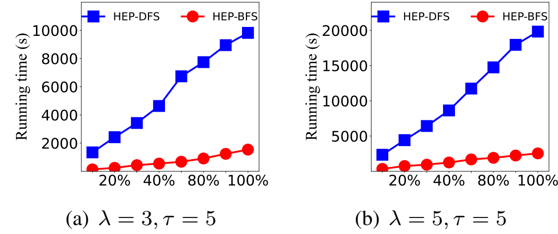
Exp-5. Running time of HEP-DFS v.s. HEP-BFS. Table III shows the running time of HEP-DFS v.s. HEP-BFS with $\lambda = 3, \tau = 5$. In addition, we also record the running time of LGR as a baseline. We can see that in all the datasets, the running time of LGR is much higher than HEP-BFS, and HEP-BFS requires approximately 10%-30% of the time of HEP-DFS on all the datasets. For example, HEP-DFS needs approximately 3690.2 seconds and 9823.1 seconds to compute all the (λ, τ)-hyperedge in WM and TVG datasets, but HEP-DFS only needs 492.4 and 1539.3 seconds, which is 13.2% and 15.6% times, respectively. This is because the pruning strategies can be easily used in Algorithm 4, and it reduces the enumerations for the possible mappings. Note that, both LGR and HEP-DFS cannot obtain results on AMZ in 1 day. The results above indicate that the pruning rules are indeed very powerful in practice.

Exp-6. Running time with varying parameters. Fig. 11 shows the running time of HEP-DFS and HEP-BFS with varying parameters on MO. We vary λ from 2 to 9 (default: 3), and τ from 3 to 10 (default: 5). Similar results can also be observed on the other datasets. As seen, HEP-BFS is faster than HEP-DFS under all parameter settings. In Fig. 11(a)-(b), the running times of HEP-DFS and HEP-BFS raise with the parameters λ and τ are increasing. And the running time of HEP-BFS raises quickly to inf (which is more than 2,000 seconds) while varying λ. These results confirm that the time complexity of HEP-BFS and HEP-DFS are linear w.r.t. τ since τ controls the termination for computing each HGED, and exponential w.r.t. λ since λ controls the candidate nodes to be added into the possible sets.



(a) vary λ (b) vary τ

Fig. 11. Running time of different algorithms on MO with varying λ, τ



(a) $\lambda = 3, \tau = 5$ (b) $\lambda = 5, \tau = 5$

Fig. 12. Scalability testings on TVG

Exp-7. Scalability. Fig. 12 shows the scalability of HEP-DFS and HEP-BFS on the TVG dataset with $\lambda = 3, \tau = 5$ and $\lambda = 5, \tau = 5$. Similar results can also be observed on the other datasets or other parameter settings. We generate ten hypergraphs by randomly picking 10%-100% of the nodes and the hyperedges, and evaluate the running times of HEP-DFS and HEP-BFS on those subgraphs. As shown in Fig. 12, the running time increases smoothly with increasing numbers of nodes or hyperedges. These results suggest that our proposed algorithms are scalable when handling large hypergraphs.

VII. CONCLUSION

In this work, we study the problem of predicting hyperedges in hypergraph. We use a concept, *Hypergraph Edit Distance*, to measure the similarity of two nodes. Specifically, we can record a *Hypergraph Edit Path* while searching the optimal edit distance, and this path enables to explain why one node is similar to another node since their neighbor structure can be edited to be isomorphic following the edit path. We propose a heuristic *DFS*-based framework which can compute the edit distance of neighbor structure for two nodes in hypergraph. Next, to predict the multiple relations, we introduce a hyperedge model in which the similarity of nodes in each links are restricted by the hypergraph edit distance. We further present an on-demand algorithm of computing *HGED*, which substantially avoids redundant computations. Finally, we conduct comprehensive experiments to show the performance of the proposed algorithms.

ACKNOWLEDGEMENT

This work was partially supported by (i) National Key R&D Program of China (Grant No. 2020AAA0108503); (ii) NSFC (Grant Nos. 62202053, 62072034, U1809206, 61932004, 62225203, U21A20516, 61732003 and U2001211); (iii) CCF-Huawei Populus Grove Fund. Rong-Hua Li and Guoren Wang are the corresponding authors of this paper.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.

References

[1] V. Martínez, F. Berzal, and J. C. C. Talavera, "A survey of link prediction in complex networks," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 69:1–69:33, 2017.

[2] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[3] B. Pandey, P. K. Bhanodia, A. Khamparia, and D. K. Pandey, "A comprehensive survey of edge prediction in social networks: Techniques, parameters and challenges," *Expert Systems with Applications*, vol. 124, pp. 164–181, 2019.

[4] I. Ahmad, M. Akhtar, S. Noor, and A. Shahnaz, "Missing link prediction using common neighbor and centrality based parameterized algorithm," *Scientific Reports*, vol. 10, p. 364, 01 2020.

[5] T. Man, H. Shen, S. Liu, X. Jin, and X. Cheng, "Predict anchor links across social networks via an embedding approach," in *IJCAI*, 2016, pp. 1823–1829.

[6] C. A. Bliss, M. R. Frank, C. M. Danforth, and P. S. Dodds, "An evolutionary algorithm approach to link prediction in dynamic social networks," *J. Comput. Sci.*, vol. 5, no. 5, pp. 750–764, 2014.

[7] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social Networks*, vol. 25, no. 3, pp. 211–230, 2003.

[8] T. Zhou, L. Lü, and Y.-C. Zhang, "Predicting missing links via local information," *The European Physical Journal B - Condensed Matter and Complex Systems*, vol. 71, pp. 623–630, 10 2009.

[9] E. Leicht, P. Holme, and M. Newman, "Vertex similarity in networks," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 73, p. 026120, 03 2006.

[10] M. Curado, "Return random walks for link prediction," *Information Sciences*, vol. 510, pp. 99–107, 2020.

[11] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *WSDM*, I. King, W. Nejdl, and H. Li, Eds., 2011, pp. 635–644.

[12] S. Han and Y. Xu, "Link prediction in microblog network using supervised learning with multiple features," *J. Comput.*, vol. 11, no. 1, pp. 72–82, 2016.

[13] C. Fu, M. Zhao, L. Fan, X. Chen, J. Chen, Z. Wu, Y. Xia, and Q. Xuan, "Link weight prediction using supervised learning methods and its application to yelp layered network," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 8, pp. 1507–1518, 2018.

[14] Z. Lu, B. Savas, W. Tang, and I. S. Dhillon, "Supervised link prediction using multiple sources," in *ICDM*, 2010, pp. 923–928.

[15] D. Luo and H. Huang, "Link prediction of multimedia social network via unsupervised face recognition," in *ACM Multimedia*, 2009, pp. 805–808.

[16] K. Li, N. Du, and A. Zhang, "A link prediction based unsupervised rank aggregation algorithm for informative gene selection," in *BIBM*, 2012, pp. 1–6.

[17] T. Kuo, R. Yan, Y. Huang, P. Kung, and S. Lin, "Unsupervised link prediction using aggregative statistics on heterogeneous social networks," in *KDD*, 2013, pp. 775–783.

[18] C. P. M. T. Muniz, R. R. Goldschmidt, and R. Choren, "Combining contextual, temporal and topological information for unsupervised link prediction in social networks," *Knowl. Based Syst.*, vol. 156, pp. 129–137, 2018.

[19] R. Abdolazimi and R. Zafarani, "Noise-enhanced unsupervised link prediction," in *PAKDD*, vol. 12712, 2021, pp. 472–487.

[20] S. Yoon, H. Song, K. Shin, and Y. Yi, "How much and when do we need higher-order informationin hypergraphs? A case study on hyperedge prediction," in *WWW*, 2020, pp. 2627–2633.

[21] T. Kumar, K. Darwin, S. Parthasarathy, and B. Ravindran, "HPRA: hyperedge prediction using resource allocation," in *WebSci*, 2020, pp. 135–143.

[22] X. Sun, H. Yin, B. Liu, H. Chen, Q. Meng, W. Han, and J. Cao, "Multi-level hyperedge distillation for social linking prediction on sparsely observed networks," in *WWW*, 2021, pp. 2934–2945.

[23] M. Zhang, Z. Cui, S. Jiang, and Y. Chen, "Beyond link prediction: Predicting hyperlinks in adjacency space," in *AAAI*, 2018, pp. 4430–4437.

[24] J. Lugo-Martinez, D. Zeiberg, T. Gaudelet, N. Malod-Dognin, N. Przulj, and P. Radivojac, "Classification in biological networks with hypergraphlet kernels," *Bioinformatics*, vol. 37, no. 7, pp. 1000–1007, 09 2020.

[25] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-13, 06 1983.

[26] H. Bunke, P. J. Dickinson, M. Kraetzl, M. Neuhaus, and M. Stettler, "Matching of hypergraphs - algorithms, applications, and experiments," in *Applied Pattern Recognition*, ser. Studies in Computational Intelligence. Springer, 2008, vol. 91, pp. 131–154.

[27] X. Zhao, C. Xiao, X. Lin, W. Wang, and Y. Ishikawa, "Efficient processing of graph similarity queries with edit distance constraints," *VLDB J.*, vol. 22, no. 6, pp. 727–752, 2013.

[28] Y. Liang and P. Zhao, "Similarity search in graph databases: A multi-layered indexing approach," in *ICDE*, 2017, pp. 783–794.

[29] X. Zhao, C. Xiao, X. Lin, Q. Liu, and W. Zhang, "A partition-based approach to structure similarity search," *Proc. VLDB Endow.*, vol. 7, no. 3, pp. 169–180, 2013.

[30] L. Chang, X. Feng, X. Lin, L. Qin, W. Zhang, and D. Ouyang, "Speeding up GED verification for graph similarity search," in *ICDE*, 2020, pp. 793–804.

[31] M. Jiang, J. Shang, T. Cassidy, X. Ren, L. M. Kaplan, T. P. Hanratty, and J. Han, "Metapad: Meta pattern discovery from massive text corpora," in *SIGKDD*, 2017, pp. 877–886.

[32] J. Shang, J. Liu, M. Jiang, X. Ren, C. R. Voss, and J. Han, "Automated phrase mining from massive text corpora," *CoRR*, vol. abs/1702.04457, 2017.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on January 23,2024 at 03:00:17 UTC from IEEE Xplore. Restrictions apply.