Efficient Meta-subgraph Instance Search over Large Heterogeneous Information Networks

LU CHEN, Swinburne University of Technology, Australia
CHENGFEI LIU, Swinburne University of Technology, Australia
RUI ZHOU, Swinburne University of Technology, Australia
JIAJIE XU , Soochow University, China
JIANXIN LI, Edith Cowan University, Australia

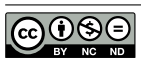
A meta-subgraph ($\mathcal{M}$), defined as an edge-unfolded subgraph consisting of a set of connected edge types in the schema of a heterogeneous information network (HIN), generalises the classical meta-path. Although prior studies have leveraged instances of special shapes defined on the schema for tasks such as cohesive subgraph discovery, similarity measurement and recommendation, no existing work has formally defined the $\mathcal{M}$-instance search problem and developed efficient algorithms dedicated to it. In this paper, we first generalise $\mathcal{M}$ queries and then systematically explore efficient algorithms dedicated to $\mathcal{M}$ instance search. For a given query $\mathcal{M}$, the vertex/edge correspondences are fixed by the schema. Thus, we do not search for embeddings; instead, we directly retrieve the corresponding instance subgraphs. This avoids generating embedding permutations, although the decision version remains NP-complete. We propose two baselines adapted from backtracking and degeneracy-ordering strategies, which scale only to small HINs. We then derive a new upper bound on the number of $\mathcal{M}$ instances, $|Ans|^*_{edge}$, a specialisation of the AGM bound, which exploits edge-type adjacency in $\mathcal{M}$ and typed edge lists. Guided by the idea of the bound, we enhance backtracking with effective pruning, achieving complexity $O^*(|Ans|^*_{edge})$. Extending and generalising this analysis to subgraphs of $\mathcal{M}$ adjacency and instances of subgraphs of $\mathcal{M}$ yields $|Ans|^*_{bag}$. By fully considering the trade-off over: 1) the cost of deriving optimum subqueries, 2) the accuracy and cost of estimating the number of instances of subqueries, and 3) the cost of materialising instances of subqueries, we propose to utilise star shape subgraphs of $\mathcal{M}$ to search $\mathcal{M}$ instances, which runs in $O^*(|Ans|^*_{star})$ and is practically much faster due to cross-star based pruning. Experiments on large real-world HINs demonstrate that our best algorithm runs up to two orders of magnitude faster than the baselines and direct AGM-based approaches.

CCS Concepts: • **Information systems** → **Network data models**; *Query optimization*; • **Theory of computation** → *Proof complexity*; **Branch-and-bound**.

Additional Key Words and Phrases: HIN, Meta-subgraph, Meta-subgraph instance

ACM Reference Format:
Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2026. Efficient Meta-subgraph Instance Search over Large Heterogeneous Information Networks. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 170 (June 2026), 26 pages. https://doi.org/10.1145/3802047

Authors' Contact Information: Lu Chen, luchen@swin.edu.au, Swinburne University of Technology, Melbourne, Victoria, Australia; Chengfei Liu, cliu@swin.edu.au, Swinburne University of Technology, Melbourne, Victoria, Australia; Rui Zhou, rzhou@swin.edu.au, Swinburne University of Technology, Melbourne, Victoria, Australia; Jiajie Xu , xujj@suda.edu.cn, Soochow University, Suzhou, Jiangsu, China; Jianxin Li, jianxin.li@ecu.edu.au, Edith Cowan University, Perth, Western Australia, Australia.



ACM 2836-6573/2026/6-ART170
https://doi.org/10.1145/3802047

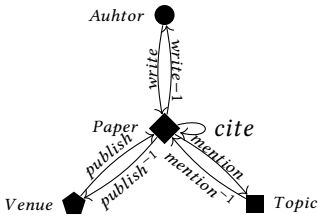


Fig. 1. Network schema

Table 1. $\mathcal{P}$ and $\mathcal{M}$ examples

ID	Query
$\mathcal{P}_1$	Author Paper Topic Paper Author
$\mathcal{P}_2$	Author Paper Venue Paper Author
$\mathcal{M}_1$	Author Paper Venue Paper Auhtor; Venue1; Author1 Paper1 Paper2 Author2; Topic Topic1

1 Introduction

HIN and schema. Heterogeneous information networks (HINs) naturally capture complex relationships among multiple types of entities and are widely used to model real-world data such as DBLP, DBpedia, and Freebase. An HIN is defined by a *network schema* $T = (\mathcal{A}, \mathcal{R})$, where $\mathcal{A}$ and $\mathcal{R}$ denote the sets of vertex and relation types, respectively, with each relation connecting two vertex types in $\mathcal{A}$. Each vertex v and edge e in the network is assigned a type via mapping functions $\phi(v) \in \mathcal{A}$ and $\psi(e) \in \mathcal{R}$. The schema provides rich semantics that enable reasoning beyond direct connections, revealing implicit yet meaningful relationships among vertices through composite relations. This schema-guided perspective leads to many fundamental analytical tools—such as similarity measures, cohesive substructure modelling, and representation learning—which have shown strong effectiveness in applications like recommendation, community search, and deep graph learning [11, 13, 19, 25, 34].

Meta-subgraph vs. meta-path. Most existing studies explore indirect relationships in HINs through a *meta-path* $\mathcal{P}$. Given the source and sink vertex types, $\mathcal{P}$ defines a composite (indirect) relation formed by a sequence of relations in $\mathcal{R}$. At the data level, a $\mathcal{P}$ instance is a concrete chain of edges following this type pattern, connecting two indirectly related vertices under semantics defined by $\mathcal{P}$. $\mathcal{P}$ is concise, but can only capture simple linear relationships.

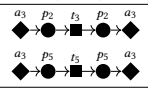
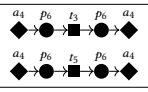
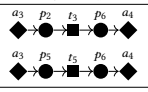
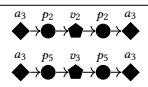
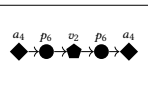
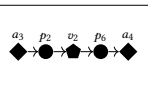
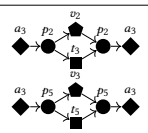
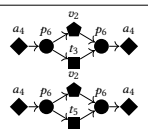
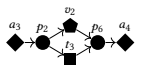
To overcome this limitation, several works generalise the concept of meta-paths to more expressive *meta-subgraphs*, such as meta-structures and meta-stars [19, 35, 36, 38, 39]. Compared with $\mathcal{P}$, meta-subgraph-based semantics enable mining richer and more complex indirect relationships among multiple vertex types defined at the schema level, thereby greatly extending the expressive power of meta-paths and benefiting many downstream applications. Those recent studies have further shown that leveraging instances of such graph-based patterns enables more in-depth analysis and reveals more meaningful information hidden in HINs.

Example 1. *Figure 1 illustrates the schema of the DBLP dataset [19, 36], and Table 1 presents two example meta-paths and a meta-subgraph. It is evident that the meta-subgraph simultaneously captures both* venue *and* topic *information while preserving the associations specified by both meta-paths. Such enhanced semantic expressiveness is critical in real applications [35, 36, 39].*

The problem. Although the expressive power of meta-subgraphs has been increasingly recognized, the problem of computing their instances has not yet been studied systematically. In this work, we formally define the meta-subgraph query and develop efficient algorithms for visiting all its instances in a given HIN.

Meta-subgraph $\mathcal{M}$. Given a network schema T, we define a *meta-subgraph* $\mathcal{M}$ as a connected edge-unfolded subgraph of T determined by edge adjacency. Specifically, $\mathcal{M}$ is formed by a subset of edge types and their adjacencies in T, thus generalizing the concept of a meta-path. Like a meta-path, $\mathcal{M}$ allows a vertex type to be *unfolded* when it is revisited through distinct edge types.

Table 2. Application example: similarity (vs $\mathcal{P}$ instance)

Query	$a_3 \to a_3$	$a_4 \to a_4$	$a_3 \to a_4$	Similarity
$\mathcal{P}_1$				$\frac{2\times2}{2+2} = 1$
$\mathcal{P}_2$				$\frac{2\times1}{2+1} \approx 0.67$
$\mathcal{M}_1$				$\frac{2\times1}{2+2} = 0.5$

This vertex unfolding is naturally implied by the edge adjacencies within $\mathcal{M}$, meaning that users only need to specify a subset of edges from T together with their adjacency relationships.

Instance of $\mathcal{M}$. Conceptually, an instance of $\mathcal{M}$ is an unfolded subgraph of the HIN that matches the structure of $\mathcal{M}$. Owing to $\mathcal{M}$ being an edge-unfolded subgraph, the notion of *matching* can be defined more clearly and concisely in terms of *edge injection* and *adjacency*. Specifically, an instance corresponds to a set of edges in H whose edge types are in one-to-one correspondence with those in $\mathcal{M}$ (equivalently, one edge is selected from each typed edge list) while preserving the adjacency relationships specified by $\mathcal{M}$.

$\mathcal{M}$ isntance search problem. Based on the above discussion, the *$\mathcal{M}$-instance search* problem aims to find all instances of a given meta-subgraph $\mathcal{M}$ in the corresponding HIN.

Application of $\mathcal{M}$ instances. As discussed, $\mathcal{M}$ is superior to $\mathcal{P}$ for defining similarity. Consider Figure 2, where we measure the similarity between a_3 and a_4. Using $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{M}_1$ in Table 1, we enumerate the instances involving a_3 and a_4 (Table 2) and apply the counting-based similarity [28]. Here, $\mathcal{P}_1$ captures authors publishing papers with a *shared topic*, and $\mathcal{P}_2$ captures authors publishing papers in a *similar venue*, yielding similarities of 1 and 0.67 (Table 2). In contrast, $\mathcal{M}_1$ merges $\mathcal{P}_1$ and $\mathcal{P}_2$ into a single pattern and captures joint semantics, authors publishing papers with the same topic and the same venue simultaneously, leading to a similarity of 0.5. This gap arises because meta-paths aggregate independent linear relations and may overestimate similarity when topic/venue signals come from different contexts, whereas $\mathcal{M}$ enforces *coupled* structural constraints for stricter semantics. Nevertheless, $\mathcal{P}$-based similarity can still be useful in simple scenarios where a simple semantic is sufficient.

Aligning with related problems. Compared with classical subgraph search problems, such as subgraph isomorphism (SubIso) and subgraph homomorphism (SubHomo), our problem is fundamentally different. First, the query $\mathcal{M}$ is defined at the schema level, rather than at the vertex level. As a result, there is no explicit query–data vertex embedding to be enumerated, which is a core requirement in both SubIso and SubHomo. As such, no embedding permutations need to be examined, and $\mathcal{M}$-instance search primarily focuses on verifying joint type-aware edge adjacencies at the data level. Moreover, $\mathcal{M}$ allows edge unfolding, a property that is absent from existing subgraph matching problems. Table 3 summarises the key differences between our problem and popular subgraph search problems.

Related techniques. To the best of our knowledge, no existing work provides a general algorithm for the $\mathcal{M}$-instance search problem. For a very special case of $\mathcal{M}$, namely the symmetric meta-path $\mathcal{P}$, state-of-the-art methods [14] leverage matrix multiplication techniques applied to one half of $\mathcal{P}$, an idea also adopted for counting specific $\mathcal{M}$ instances in [39]. However, these techniques are

Table 3. Comparison with subgraph search problems

Method	Input data	Query	Search semantic	Output	Complexity	Special case
SubHomo	Graph (G)	Subgraph (Q) consisting of distinct vertices	$f: V(Q) \to V(G)$, non-injective, adjacency-preserving	All SubHomo embeddings	$\lvert V(G)\rvert^{\lvert V(Q)\rvert}$	Triangle k-clique
Join-based SubHomo			Join results of join encoded Q	All distinct joined results	AGM bound	
SubIso			$f: V(Q) \to V(G)$, inj., adj. pres.	All SubIso embeddings	$\lvert V(G)\rvert^{\underline{\lvert V(Q)\rvert}}$	
$\mathcal{M}$ instance search	HIN: Graph with network schema	$\mathcal{M}$ at the **schema level** (duplicated vertex type is allowed)	Definition 2.2: data-level edges with edge types pres. adj.	All distinct $\mathcal{M}$ instances	AGM-based bound	Meta-path Meta-structure

designed for specific structural patterns and cannot be directly applied to the general $\mathcal{M}$ search problem. The backtracking-based method proposed in [36] explores subgraphs guided by $\mathcal{M}$, using level-wise adjacency information to prune the search space and derive instances. Nevertheless, it remains unclear how this approach can handle $\mathcal{M}$ without a well-defined level structure. Therefore, novel techniques are required to efficiently address the general $\mathcal{M}$-instance search problem.

Hardness and technical challenges. To establish a realistic performance objective, we first conduct a hardness analysis. We prove that the decision version of the $\mathcal{M}$-instance search problem is NP-complete via a non-trivial reduction from the k-clique problem. This result implies that exhaustive backtracking may explore exponentially many combinations. The first challenge is how we can explore vertex combinations guided by adjacency to efficiently derive valid $\mathcal{M}$ instances. The second challenge is how to systematically apply pruning driven by $\mathcal{M}$'s structural constraints so that the overall exploration cost can be effectively bounded and scales with the number of valid instances rather than the total number of vertex combinations.

Our solution. To address the first challenge, we annotate $\mathcal{M}$ into $\mathcal{M}'$ by assigning unique identifiers to repeated vertex types using edge-type adjacency in $\mathcal{M}$, resolving ambiguity. $\mathcal{M}'$ is equivalent to $\mathcal{M}$ but represented as an annotated vertex-type adjacency list. Based on $\mathcal{M}'$, we conceptually refine G into a restricted subgraph $\mathcal{H}_{\mathcal{M}'}$, where some vertex types may be duplicated. Using $\mathcal{H}_{\mathcal{M}'}$, we develop two baselines: one follows structure-based pruning as in subgraph isomorphism, and the other adopts degeneracy ordering. Both, however, still run exponentially with trivial bounds.

To address the second challenge, we exploit edge-type adjacency information in the schema to further reduce candidate sets based on partial results. Intuitively, this ensures that each remaining candidate has feasible neighbours required by $\mathcal{M}$, thereby preventing invalid expansions early. We rigorously prove that the resulting algorithm runs proportionally to the worst-case number of $\mathcal{M}$ instances, with the bound derived using Finner's inequality. This bound matches the well-known AGM bound under our problem configuration, denoted as $|Ans|^*_{edge}$.

We further generalise the above analysis from individual edge sets to bags of subqueries, leading to the $|Ans|^*_{bag}$ bound. This enables a more general search strategy by decomposing $\mathcal{M}'$ into smaller subqueries with overlaps, which provide additional filtering opportunities and can yield tighter bounds when instances of subgraph shapes can be computed efficiently. However, deriving optimal bags (NP-hard in general) and materialising the results of subqueries are non-trivial. In particular, we consider star-shaped subqueries, whose instances can be obtained at negligible cost, to derive both improved bounds and faster algorithms. We show that selecting the optimal set of star shapes can be solved in polynomial time. This further improves the overall performance.

Contributions. Our principal contributions are below.

- We systematically study the meta-subgraph search problem.
- We propose non-trivial pre-pruning and baselines as foundations.
- We present a backtracking algorithm that explores edge-type-adjacency (schema-level) and local-neighbour (data-level) reductions, thereby matching the AGM edge bound.

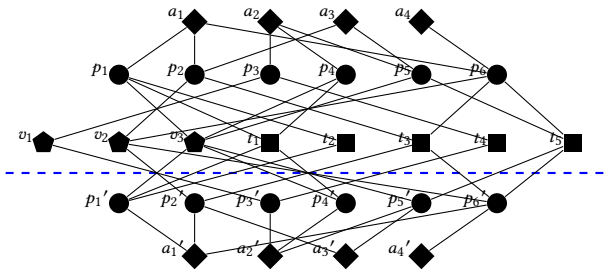


Fig. 2. An instance of HIN data and expanded data for $\mathcal{M}$

- We further generalise the AGM edge bound to the AGM bag bound and present star-shaped bagging strategies to further improve efficiency, which is faster for large HINs.
- We conduct extensive experimental studies to show the efficiency and effectiveness of our proposed techniques and algorithms.

2 Preliminary and problem formulation

We introduce related concepts first. The frequently used notations in this paper are summarized in Table 4.

HIN. An HIN is defined as a graph $G = (V, E)$ with a vertex type mapping function $\phi : V \to \mathcal{A}$ and an edge type mapping function $\psi : E \to \mathcal{R}$, in which every $v \in V$ has a vertex type $\phi(v) \in \mathcal{A}$ and every $e \in E$ has an edge type $\psi(e) \in \mathcal{R}$.

Network schema. The network schema is a meta template for an HIN $G = (V, E)$ with the vertex type mapping $\phi : V \to \mathcal{A}$ and the edge type mapping function $\psi : E \to \mathcal{R}$, which is a directed graph defined over vertex types $\mathcal{A}$, with edges as relations from $\mathcal{R}$, denoted as $T = (\mathcal{A}, \mathcal{R})$.

Meta-path. A meta path $\mathcal{P}$ is a path defined on the network schema T_G, and is denoted in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \ldots \xrightarrow{R_{i-1}} A_i$, which defines a composite relation $R = R_1 \circ R_2 \circ \ldots \circ R_{i-1}$ between type A_1 and A_i, where $\circ$ denotes the composition operator on relations.

Instance of A **and** R. Given a vertex type $A \in \mathcal{A}$ (edge type $R \in \mathcal{R}$), a vertex v (an edge (u, v)) is an instance of A (R), if and only if $\phi(v) = A$ ($\psi(\phi(u), \phi(v)) = R$).

2.1 Problem formulation

In this section, we first extend the concept of meta-path to meta-subgraph $\mathcal{M}$, allowing users to express more complicated query requirements at the schema level. Then, we formally define the problem of $\mathcal{M}$ instance search.

We first introduce the following observation.

Observation 1. *Many HIN instances have concise schemas and implicit vertex or edge types exist, which makes a vertex/ an edge exhibit multiple types. For instance, in the widely used DBLP schema, the relation author-write-paper implicitly defines the relation paper-written_by-author, which means the author also has an implicit semantic role.*

Next, we discuss extending T by incorporating the implicit semantic meaning as follows.

Relation expansion For each undirected edge $(A_i, A_j) \in \mathcal{R}$ ($A_i \neq A_j$), its reversal (A_j, A_i) is included into T if $(A_j, A_i) \notin \mathcal{R}$.

Based on the expansion, we are ready to propose a new query pattern defined on the schema level.

Table 4. Notations

Notation	Definition
$\mathcal{A}$	the set of all vertex types in an HIN
$A, A_i, A_j, X_i, \ldots$	a vertex type in $\mathcal{A}$
$\mathcal{M}, \mathcal{M}'$	a meta-subgraph, $\mathcal{M}$ annotated with id
$V(A), E((A_i, A_j))$	A type vertices, (A_i, A_j) type edges
$\mathcal{H}_{\mathcal{M}'}$	vertex-type-expanded HIN w.r.t. $\mathcal{M}'$
$\mathcal{R}_{\mathcal{M}'}, \mathcal{A}_{\mathcal{M}'}$	edge, vertex types in $\mathcal{M}'$
$R_{(X_i,X_j)}$	edge type consisting of X_i and X_j
$N(u, A)$	neighbours of u with vertex type A in graph
$N(A, \mathcal{M}')$	neighbours of vertex type A in $\mathcal{M}'$
$C[\cdot]$	candidate vertex set indexed by vertex type
$\mathcal{B}[A]$	bags involving A

Definition 2.1. **Meta-subgraph $\mathcal{M}$**: Given an HIN schema $T = (\mathcal{A}, \mathcal{R})$, $\mathcal{M} = (\mathcal{R}_{\mathcal{M}}, e\_adj_{\mathcal{M}})$ is formulated as follows:

- Edge-type subset: $\mathcal{R}_{\mathcal{M}} \subseteq \mathcal{R}$
- Provided edge adjacency: $e\_adj_{\mathcal{M}} \subseteq \mathcal{R}_{\mathcal{M}} \times \mathcal{R}_{\mathcal{M}}$
  - $(R_i, R_j) \in e\_adj_{\mathcal{M}}$ implies that R_i and R_j share on vertex type, but the converse does not necessarily hold.
- Connectivity: For any two R_i and R_j, either (R_i, R_j) is in $e\_adj_{\mathcal{M}}$, or there exists a sequence $(R_i = R_{s_0}, R_{s_1}, \ldots, R_{s_k} = R_j)$ such that $(R_{s_{t-1}}, R_{s_t}) \in e\_adj_{\mathcal{M}}$ $\forall\ 1 < t \leq s_k$.

Why edge-type adjacency. Readers may question why we define the meta-subgraph $\mathcal{M}$ based on edge-type adjacency rather than directly on vertex-type connectivity. The rationale is that, in heterogeneous schemas, a single vertex type can participate in multiple relations and thus play different semantic roles across contexts. Consequently, a vertex type may reappear in $\mathcal{M}$ when it connects distinct relation types. This phenomenon, referred to as edge unfolding, induces type duplication, in which the same vertex type occurs multiple times but occupies different structural positions within the pattern. By defining $\mathcal{M}$ through its edge-type adjacency structure, we elegantly capture these unfolded roles without introducing artificial vertex identifiers or enforcing a global one-to-one correspondence between schema types and their occurrences in $\mathcal{M}$. Moreover, this edge-centric representation also aligns well with $\mathcal{M}$ instance that will be defined later, where both the mapping and the preservation of structure are expressed at the level of edges.

Why only a subset of edge types. First, allowing the same relation type to appear multiple times would imply duplicated semantics (e.g., repeatedly using the same *author–paper* relation), which relates more to defining cohesiveness at the instance level rather than at the schema level, thereby contradicting the essence of a schema that abstracts unique relationship types. Second, this restriction avoids redundant mapping permutations among identical relations, making instance matching more efficient and well-defined. It also ensures that each $\mathcal{M}$ captures clear and non-overlapping semantics, making the resulting instances more interpretable and practical for downstream applications such as similarity measurement, community discovery, and representation learning. Later, we will discuss how our proposed algorithm can process border query patterns defined on T.

Definition 2.2. **Instance of $\mathcal{M}$**. Given G with T and $\mathcal{M} = (\mathcal{R}_{\mathcal{M}}, e\_adj_{\mathcal{M}})$ serving as a query, an $\mathcal{M}$ instance is collection of edges $H = \{e = (u, v) | \psi(\phi(u), \phi(v)) \in \mathcal{R}_{\mathcal{M}}\}$ if the following hold.

- Edge injectivity: each edge $(u, v) \in H$ corresponds to a distinct edge type $\psi(\phi(u), \phi(v)) \in \mathcal{R}_{\mathcal{M}}$.
- Adjacency preservation: for every adjacent edge types $(R_i, R_j) \in e\_adj_{\mathcal{M}}$, their data instances share a common endpoint whose type matches the shared schema vertex type.

We are ready to formally introduce the problem we studied.

Problem 1. *Given an HIN G with T, a query $\mathcal{M}$, finding all $\mathcal{M}$ instances for G.*

The following lemma shows the hardness of our problem.

Lemma 2.3. *The problem of deciding whether an $\mathcal{M}$ instance exists is NP-complete.*

Proof. First, the problem is in NP. Given H defined above, checking injectivity (edge types) and joint adjacencies is polynomial.

Second, we reduce k-clique problem (when $k > 3$) to an instance of our problem. Given k, decide whether $G = (V, E)$ contains a clique of size at least k. The construction is as follows. The schema of the instance of our problem contains $\mathcal{A} = \{A_1, \dots A_k\}$ and $\mathcal{R} = \{(A_i, A_j) | 1 \le i < j \le k\}$. The HIN data instance is as follows. For each vertex type, copy V to be that type. For each typed relation $R = (A_i, A_j) \in \mathcal{R}$ $(i < j)$, $E(R) = \{(u, v) \in V(A_i) \times V(A_j) | \{u, v\} \in E_G\}$. Let the query $\mathcal{M}$ be the created schema. This construction is in p-time clearly.

If G has a k-clique C, an $\mathcal{M}$ instance exists. Let $C = \{c_1, \dots, c_k\} \subseteq V$. For each $(i < j)$, choose $(c_i, c_j) \in E((A_i, A_j))$, which leads to $H = \{(c_i, c_j)\}_{i<j}$ be an valid $\mathcal{M}$ instance. On the other hand, if an $\mathcal{M}$ instance $H = \{(u_{i,j}, v_{i,j}) \in R(A_i, A_j) | 1 \le i < j \le k\}$ exists, for any i, (A_i, A_j) and (A_i, A_l) are adjacent in the schema, so adjacency preservation requires their data edges to share the same endpoint of type A_i. As such, there exists a unique vertex $c_i \in V$ such that $(u_{i,j}, v_{i,j}) = (c_i, c_j)$ for all $i < j$. Collecting such vertices leads to $C = \{c_1, \dots, c_k\} \subseteq V(G)$, which is a k-clique. □

Remark. For conciseness, we treat incoming and outgoing (or typed) edges uniformly by combining them for the remainder of this paper, as their directionality (or types) can be easily handled during implementation.

3 Warm-up: search space, pre-prunings and baselines

The hardness analysis in the previous section indicates that unless $P = NP$, there exists no polynomial-time algorithm for solving the $\mathcal{M}$-instance search problem in general. Consequently, in this section, we introduce two backtracking-based baselines, which systematically explore vertex combinations constrained by adjacency relations to visit all instances of $\mathcal{M}$.

3.1 Search space and pre-pruning

We begin by constructing the search space, i.e., the actual graph data that may contain instances of $\mathcal{M}$. This structure, called the type-expanded HIN $\mathcal{H}_{\mathcal{M}'}$, defines all valid candidate vertices and edges for $\mathcal{M}$-instance search.

From edge-type adjacency to vertex-type adjacency. We first construct a vertex-to-incident-edge list from $\mathcal{M}$, recording, for each vertex type in the schema, all edge types incident to it. If a vertex type appears multiple times as an entry in this list—meaning it participates in multiple distinct edge types—it is assigned a distinct *role identifier* for each occurrence. This ensures that every vertex type occurrence in $\mathcal{M}$ is explicitly represented. We then traverse the edge-adjacency structure of $\mathcal{M}$, connecting vertex-type roles according to the corresponding edge types, thereby obtaining a role-indexed vertex-type adjacency representation, denoted by $\mathcal{M}'$. The transformation runs in linear time $O(|\mathcal{R}_{\mathcal{M}}| + |e\_adj_{\mathcal{M}}|)$ and produces an unambiguous structure that preserves the edge-type connectivity of $\mathcal{M}$, serving as the foundation for constructing $\mathcal{H}_{\mathcal{M}'}$.

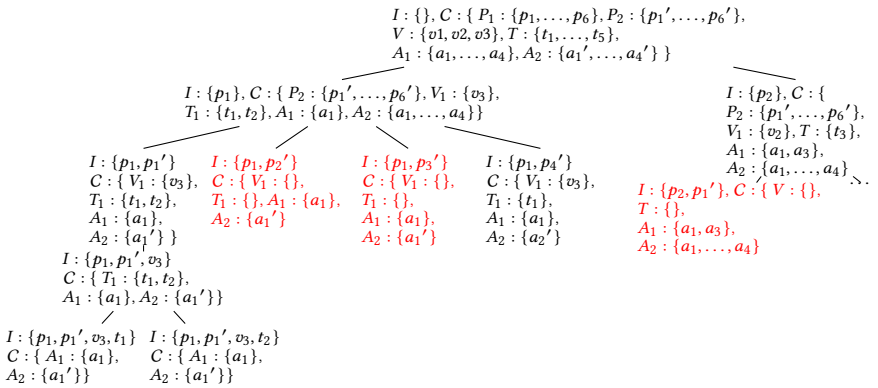


Fig. 3. Part of the backtracking tree

Example 2. *Considering $\mathcal{M}_1$ in Table 1, the black words indicate vertex types in the input. Based on the edge adjacency, Author serves as the entry of edge adjacency twice and therefore it becomes Auhtor1 and Author2. The same applies to Paper. To maintain naming consistency, other vertex types serving as one entry for some edge adjacency are also appended with* 1. *The gray colored vertex types in Table 1 are used for vertex-type adjacency (vertex types in $\mathcal{M}_1{}'$).*

Expanded HIN $\mathcal{H}_{\mathcal{M}'}$. The expanded graph $\mathcal{H}_{\mathcal{M}'}$ is constructed by aligning each vertex-type occurrence $A_i \in \mathcal{A}_{\mathcal{M}'}$ with all data vertices of type $V(A_i)$ in G, and connecting them according to the corresponding typed relations in G, where ids are ignored. By construction, $\mathcal{H}_{\mathcal{M}'}$ preserves all vertex and edge type constraints defined by $\mathcal{M}'$. Hence, any data subgraph that satisfies the type and adjacency requirements of $\mathcal{M}$ must be fully contained in $\mathcal{H}_{\mathcal{M}'}$. In other words, if an $\mathcal{M}$-instance exists, it necessarily appears as a subgraph of $\mathcal{H}_{\mathcal{M}'}$, making $\mathcal{H}_{\mathcal{M}'}$ a complete and lossless search space for$\mathcal{M}$ instance discovery.

Example 3. *Figure 2 illustrates the expanded HIN constructed for $\mathcal{M}_1{}'$ shown in Table 1. The input HIN network without expansion is shown above the blue line. Since in $\mathcal{M}_1{}'$ there are Auhtor1 and Auhtor2, the author type vertices appear twice in Figure 2, i.e., $a_1,\ldots,a_4$ and $a_1',\ldots,a_4'$ (the same for the paper type vertices). The edge set between the renamed vertex types mimics the set of edges between the original vertex types. As such $\mathcal{H}_{\mathcal{M}_1{}'}$ is generated, say, all vertices and edges in Figure 2.*

Pre-pruning. We are now ready to reduce the search space by pruning vertices in $\mathcal{H}_{\mathcal{M}'}$. The pruning progressively removes any vertex that violates its *local neighbour requirement*, defined below.

Definition 3.1. Given a vertex $u \in \mathcal{H}_{\mathcal{M}'}$, its *local neighbour requirement* is satisfied if $\forall A \in N(\phi(u), \mathcal{M}')$, $N(u, \mathcal{H}_{\mathcal{M}'}, A) \neq \emptyset$, where $N(\phi(u), \mathcal{M}')$ denotes the set of adjacent vertex types of $\phi(u)$ contained in $\mathcal{M}'$, and $N(u, \mathcal{H}_{\mathcal{M}'}, A)$ denotes the neighbours of u of type A in $\mathcal{H}_{\mathcal{M}'}$.

The pruning process proceeds iteratively, removing vertices that violate the condition stated above. It runs in $O(|E(\mathcal{H}_{\mathcal{M}'})|)$ time since each vertex is removed at most once, and upon removal, only its neighbours require updates.

After pre-pruning, we safely refine the data graph G to a compact search space $\mathcal{H}_{\mathcal{M}'}$ in which all vertices satisfy the local neighbour requirement. $\mathcal{H}_{\mathcal{M}'}$ serves as the input to our proposed algorithms, providing a more focused search space that preserves all valid $\mathcal{M}$-instances.

Algorithm 1: Baseline: btbase($\mathcal{H}_{\mathcal{M}'}$, $\mathcal{M}'$)

$C \leftarrow \{\{\emptyset\} \ldots, \{\emptyset\}\}$;// indexed by vertex types
foreach $A \in \mathcal{A}(\mathcal{M}')$ **do** $C[A] \leftarrow V(A)$;
let O be an order for $\mathcal{A}(\mathcal{M}')$, $I \leftarrow \emptyset$, let A be the first vertex type in the order;
foreach $v \in C[A]$ **do**
    $I \leftarrow I \cup \{v\}$, $C' \leftarrow C$, $C'[A] \leftarrow \emptyset$;
    /* refine candidate set */
    foreach $C[A'] \in C$ *s.t.* A' *incident to* A *in* $\mathcal{M}'$ *and appering later than* A *in* O **do**
        $C' \leftarrow C$, $C'[A'] \leftarrow C[A'] \cap N(v)$;
        if $C'[A'] == \emptyset$ **then** $I \leftarrow I \setminus \{v\}$, continue outer;
    BTSearching(I, C'), $I \leftarrow I \setminus \{v\}$;
Procedure BTSearching(I, C)
    if $|I| == |\mathcal{A}(\mathcal{M})|$ **then** visit I;
    let A be the vertex type at the $|I| + 1$ position in O;
    /* refine candidate set */
    foreach $v \in C[A]$ **do**
        $I \leftarrow I \cup \{v\}$, $C' \leftarrow C$, $C'[A] \leftarrow \emptyset$;
        foreach $C[A'] \in C$ *s.t.* A' *incident to* A *in* $\mathcal{M}'$ *and appearing later than* A *in* O **do**
            $C'[A'] \leftarrow C[A'] \cap N(v)$;
            if $C'[A'] == \emptyset$ **then** $I \leftarrow I \setminus \{v\}$, continue outer;
        BTSearching(I, C'), $I \leftarrow I \setminus \{v\}$;

3.2 Schema level order fixed approach

In this section, we present a search framework that adapts core ideas from state-of-the-art subgraph isomorphism algorithms. Following the same principle, we first fix a total search order based on the query structure and then perform a backtracking search. Thanks to the construction in the previous section, the framework can now be seamlessly adapted using $\mathcal{M}'$ and $\mathcal{H}_{\mathcal{M}'}$, as detailed below.

Intuition and key idea. Since the problem is NP-hard, a naive backtracking search would need to explore all possible vertex combinations over the typed vertex sets defined in $\mathcal{M}'$, verifying both vertex and edge type consistency and the overall adjacency structure. However, this can be done much more efficiently. Whenever a vertex is included in a partial result, we immediately prune the remaining candidate sets by enforcing the edge-type–aware adjacency constraints specified by $\mathcal{M}'$. This progressive refinement greatly reduces the search space explored during backtracking. Moreover, when a complete result of size $|\mathcal{A}_{\mathcal{M}'}|$ is generated, there is no need for an additional structural verification, as the search process inherently guarantees all required adjacency constraints.

The algorithm. Algorithm 1 outlines the main steps of the backtracking framework and is largely self-explanatory. Note that we do not need to explicitly maintain the candidate vertices for types that have not yet been reduced by adjacency-based filtering in the intermediate result set I. Instead, these candidates can be retrieved directly from the data graph $\mathcal{H}_{\mathcal{M}'}$ on demand when their corresponding vertex types are subject to adjacency-based reduction.

Time complexity. The time complexity is $O\Big(|C(A_1)| \cdot d_{\max}^{|\mathcal{A}(\mathcal{M}')|-1} \cdot d_{\max} \cdot d(\mathcal{M}')\Big)$, where $|C(A_1)|$ is the number of candidate vertices for the first vertex type A_1 in the search order O, and $d(\mathcal{M}')$ denotes the maximum degree of $\mathcal{M}'$. The first two factors correspond to the cost of enumerating all possible vertex combinations, i.e., $\prod_{A \in \mathcal{A}(\mathcal{M}')} |C(A)|$, while the last two factors capture the per-recursion cost determined by adjacency checks during backtracking.

Example 4. *Let us continue with Example 3. We further show backtracking tree for $\mathcal{M}_1{}'$ on $\mathcal{H}_{\mathcal{M}_1{}'}$. The search order is $O = (P_1, P_2, V_1, T_1, A_1, A_2)$, where P_1 is the abbreviation of Paper1, similarly for others in O. Following the order, as shown in the first recursion state of the left branch in Figure 3, when p_1 is included in I, since P_1 is adjacent to A_1, V_1 and T_1, these types of vertices in C are then reduced (via intersection operations) according to p_1 neighbours of these types. Similar operations are applied for other recursion states. Although C is gradually reduced during the search, many recursion states still need to be explored that do not lead to valid $\mathcal{M}$ instances (highlighted in red). For brevity, recursion states that obviously lead to a valid instance (i.e., with exactly one vertex per vertex type) are omitted.*

3.3 Sparsity aware data level ordering

Motivation. Large HINs are generally sparse and exhibit low degeneracy. Leveraging a degeneracy ordering can therefore significantly improve performance, enabling search algorithms with lower parameterised time complexity. However, directly applying degeneracy-based traversal introduces several challenges. First, while degeneracy ordering effectively bounds the number of one-hop neighbours, our problem involves matching patterns that span multiple hops. Second, enforcing degeneracy order at the data level may easily conflict with the predefined schema-level order used to ensure structural consistency. Thus, the key question is how to exploit low graph degeneracy while preserving the correctness and completeness of $\mathcal{M}$-instance enumeration.

Overview. We propose an *adaptive schema-ordering search algorithm* that integrates degeneracy-based traversal at the data level while respecting the structural constraints defined by the schema. The algorithm dynamically aligns the data-level exploration with an adaptive schema order, combining pruning efficiency from degeneracy with correctness guarantees for type consistency.

Algorithm and correctness. Algorithm 2 follows a depth-first traversal strategy and maintains the following key operations. Initially, the candidate set includes all data vertices for each vertex type in $\mathcal{A}_{\mathcal{M}'}$. When a vertex u in the degeneracy order is selected to I, the size of its neighbours of adjacency types in $\mathcal{M}'$ is immediately reduced to the core number of u (bounded by δ). During the search, for each vertex type adjacent to any vertex type already included in the current I, its candidate set is refined based on adjacency constraints in $\mathcal{H}_{\mathcal{M}'}$. This ensures that every vertex added to I satisfies all adjacency requirements observed so far. Moreover, once a vertex v is included in I, all other vertices of the same type $\phi(v)$ are excluded from subsequent recursive calls, guaranteeing that each vertex type in $\mathcal{M}'$ contributes exactly one vertex to I. As such, when the recursion reaches depth $|I| = |\mathcal{A}_{\mathcal{M}'}|$, I forms a valid $\mathcal{M}'$-instance. By construction, this procedure enumerates all $\mathcal{M}$-instances without duplication and without violating adjacency constraints, while benefiting from the reduced candidate space induced by low degeneracy.

Optimisations. Since the data-level ordering only affects candidate reductions in lines 7–8 of Algorithm 1, the specific order used in Line 14 does not influence correctness. To further improve efficiency, we reorder the remaining vertex types in line 14 by ascending candidate set size, giving priority to those with the fewest remaining candidates. With this ordering, once the early vertex type becomes exhausted, the recursion can be safely terminated early (lines 5 and 21), as no complete instance can be formed thereafter. This heuristic both reduces unnecessary branching during backtracking and allows the degeneracy parameter δ to more directly govern the effective time complexity.

Algorithm 2: degbase($\mathcal{H}_{\mathcal{M}'}$,$\mathcal{M}'$)

```
1  O vertices in H_{M'} in a degeneracy order;
2  build forward neighbours for N^+(v) ∀v ∈ O;
3  C ← sets of vertices of type A, ∀ A ∈ 𝒜(M');
4  foreach v ∈ O in order do
5      if early termination condition is satisfied then return;
6      I ← {v}, C' ← C ;
7      foreach A ∈ 𝒜(M') s.t. A incident to φ(v) in M' do
8          C'[A] ← C[A] ∩ N^+(v);
9          if C'[A] == ∅ then I ← I \ {v}, continue outer;
10     C'[φ(v)] ← ∅;
11     AdaptiveBT(I, C'), I ← I \ {v};
12 Procedure AdaptiveBT(I, C)
13     if |I| == |𝒜_{M'}| then visit I;
14     foreach v ∈ C do
15         C' ← C, I ← I ∪ {v};
16         foreach A ∈ 𝒜(M') s.t. A incident φ(v) in M' do
17             if R(φ(v), A) ∈ ℛ(M') then
18                 C'[A] ← C[A] ∩ N(v);
19             If C'[A] == ∅ then I ← I \ {v}, continue outer;
20         C'(φ(v)) ← ∅;
21         if C' cannot satisfy early stop condition then
22             AdaptiveBT(I, C'), I ← I \ {v};
```

Time complexity. With the optimisation, the cost is dominated by $O\left(|V(\mathcal{H}_{\mathcal{M}'})|\,\delta\, d_{\max}^{|\mathcal{A}(\mathcal{M}')|-2}\right)$ recursive calls, where δ is the graph degeneracy and $d_{\max}$ denotes the maximum degree in $\mathcal{H}_{\mathcal{M}'}$. Each recursive call can be further bounded by $d_{\max}$, corresponding to the cost of enumerating neighbours in adjacency checks.

3.4 Discussion

The two baselines introduced so far follow classical backtracking principles. The first baseline explores combinations by directly checking edge adjacencies through schema-guided ordering, while the second uses degeneracy-aware order at the data level with dynamic schema ordering.

Although the two baselines are correct, their performance is limited by the combinatorial nature of recursive exploration. To mitigate this, we adopt a more holistic strategy that maximises candidate filtering by leveraging overlapping intermediate results. The key idea is to decompose $\mathcal{M}$ into smaller, interrelated sub-patterns that can be evaluated efficiently; their interrelations enable strong cross-filtering of candidates. This naturally leads to our jigsaw framework, which may substantially improve the time complexity.

4 Jigsaw based approach

In this section, we introduce a jigsaw-based framework that builds on the divide-and-conquer principle to improve the efficiency of $\mathcal{M}$-instance search. Rather than explicitly solving and merging

subproblems, the Jigsaw approach decomposes the query $\mathcal{M}'$ into smaller, interrelated sub-patterns to guide candidate pruning during the search.

High-level idea and objectives. Since $\mathcal{M}'$ is typically small, we can afford to invest additional effort in optimising the search strategy. The key idea is to decompose $\mathcal{M}'$ into smaller, interrelated sub-queries such that (1) each sub-query can be efficiently evaluated with non-dominating cost, and (2) their partial results can be leveraged to progressively filter candidates, thereby reducing the overall search complexity.

Our main result. To achieve the above objective, we first propose filtering candidates using typed edge lists, and rigorously prove that, when combined with our backtracking algorithm, the time complexity aligns with the well-known AGM bound under our setting, denoted by $|Ans|^*_{edge}$.

To further enhance efficiency, we extend this idea by filtering candidates through *instances of subgraphs* of $\mathcal{M}'$, yielding a more general bound $|Ans|^*_{bag}$. However, generating the optimal bag set is NP-hard, and computing or storing subgraph instances can incur substantial overhead, limiting its practicality.

To address this, we focus on *star-shaped subgraphs* of $\mathcal{M}'$, whose instances can be computed in linear time without materialisation. We further show that selecting optimal star bags is polynomially solvable, leading to our final algorithm that achieves $O^*(|Ans|^*_{star})$ complexity while remaining highly efficient in practice.

Below, we show detailed techniques section by section.

4.1 Jigsaw baseline: filtering by incident edges

In this section, we propose using adjacency in $\mathcal{M}'$ to further filter candidate sets.

More filtering by adjacency. The intuition is to treat typed edge lists as partial results and progressively refine them as the search progresses. At each recursive step, we restrict attention to those edge lists that are relevant to the current partial result I. When selecting a vertex u of type A as the next candidate to extend I, we further verify whether u has valid neighbours for all incident edge types of A in $\mathcal{M}'$. That is, for every $A' \in N(A, \mathcal{M}')$, the vertex u must appear in all edge lists of type (A, A') in $\mathcal{H}_{\mathcal{M}'}$ filtered by I. This filtering resembles the pre-pruning phase but is applied dynamically within the context of the current partial result, providing a more substantial reduction of candidate vertices.

The algorithm. By applying the above pruning strategy, we present Algorithm 3. Algorithm 3 progressively selects candidate vertices that satisfy adjacency constraints across all incident edge types in $\mathcal{M}'$, thereby maintaining local structural consistency throughout the search. Note that $\mathcal{H}_{\mathcal{M}'}$ already satisfies the valid-neighbour condition after pre-pruning; therefore, Algorithm 3 operates similarly to Algorithm 1 for the first few steps. During recursion, line 4 further enforces the valid-neighbour condition with respect to the current partial result I, ensuring that each recursive call only considers candidates that are not only consistent with all relevant incident edge types but also have corresponding valid neighbours in the local subgraph up to the time. Since the filtering continuously eliminates infeasible vertices under these constraints, the number of feasible local adjacency configurations tightly bounds the total number of operations.

Example 5. *Based on Example 4, we further show how Algorithm 3 avoids visiting the red recursive states that cannot lead to a valid $\mathcal{M}$ instance. Let us focus on the first two red states induced by selecting p_2' and p_3'. After including p_1 in I and before Algorithm 3 begins to add a P_2 type vertex to I, it further prunes vertex types adjacent to V_1 and T_1, since the candidates for V_1 and T_1 in C have been reduced, as shown in Example 4. Since P_2 type is adjancet to V_1 and T_1, P_2 can be further reduced based on the fact $V_1 = \{v_3\}$ and $T_1 = \{t_1, t_2\}$. As such, for the first recursion state in the left branch Algorithm 3 further reduces P_2 type of vertices in C, say, the union of P_2 neighbours of v_3, P_2 neighbours of t_1,*

and P_2 neighbours of t_2, which excludes p'_2 and p'_3. Algorithm 3 does not enter such recursive states, achieving higher efficiency than Algorithm 1.

Bounding the total number of recursions. Let us focus on when Algorithm 3 works at the $|\mathcal{A}_{\mathcal{M}'}| - 1$ depth with partial result I, where the candidate vertex type is X_m. By the algorithm, the number of recursions for $|\mathcal{A}_{\mathcal{M}'}|$ depth can be bounded as follows:

$$\sum_{v_m \in X_m} \prod_{R_{(X_m,X_{j'})} \in R(X_m)} 1_{R_{(X_m,X_{j'})}(v_m,v_{j'})}, \tag{1}$$

where $R(X_m)$ denotes edge types incident to X_m. Equation 1 counts how many candidates $v_m \in X_m$ are compatible with I: i.e., for every neighbour $X_{j'}$ of X_m, the typed edge $(v_m, v_{j'})$ exists and $v_{j'} \in I$, which is precisely what the new pruning introduced in this section does. Notice that edges incident to X_m could be in or out edges. Rigorously, we may need to split edges to be $R_{(X_m,X_{j'})}$ and $R_{(X_{j'},X_m)}$. For the purpose of brevity, we write them all as $R_{(X_m,X_{j'})}$.

Before deriving the bound, let us briefly discuss Finner's inequality.

Lemma 4.1 (Finner's Inequality [12]). *Let $\mathcal{F}$ be a finite family of subsets of an index set $\mathcal{V}$, and let $\{f_F\}_{F\in\mathcal{F}}$ be nonnegative measurable functions $f_F : \prod_{v\in F} X_v \to \mathbb{R}_{\geq 0}$. For nonnegative weights $\boldsymbol{\alpha} = \{\alpha_F\}_{F\in\mathcal{F}}$ satisfying*

$$\sum_{F \ni v} \alpha_F \geq 1 \quad \textit{for every } v \in \mathcal{V},^{1}$$

we have

$$\int_{\prod_{v\in\mathcal{V}} X_v} \prod_{F\in\mathcal{F}} f_F(x_F)\, d\mu(x) \;\leq\; \prod_{F\in\mathcal{F}} \left(\int_{\prod_{v\in F} X_v} f_F(x_F)^{1/\alpha_F}\, d\mu_F(x_F) \right)^{\alpha_F}.$$

Intuition. This inequality generalises Hölder's inequality to families of functions defined on overlapping coordinate subsets. Each variable v contributes to the integral through all functions f_F containing it, weighted by α_F, ensuring the overall exponent condition $\sum_{F\ni v} \alpha_F \geq 1$.

Let us apply Finner's inequality to Equation 1, we have

$$\sum_{v_m \in X_m} \prod_{R_{(X_m,X_{j'})} \in R(X_m)} 1_{R_{(X_m,X_{j'})}(v_m,v_{j'})} \leq \tag{2}$$

$$\prod_{R_{(X_m,X_{j'})} \in R_{x_m}} \sum_{v_m \in X_m} 1_{R_{(X_m,X_{j'})}(v_m,v_{j'})}^{\frac{1}{\alpha_{R_{(X_m,X_{j'})}}}\alpha_{R_{(X_m,X_{j'})}}} \tag{3}$$

subject to $\sum_{R_{(X_m,X_{j'})}\in R(X_{j'})} \alpha_{R_{(X_m,X_{j'})}} \geq 1$ and $\alpha_{R_{(X_m,X_{j'})}} \geq 0$. Since $1^{\frac{1}{\alpha_{R_{(X_m,X_{j'})}}}}$ is 1, we can omit this exponent, which becomes $\prod_{R_{(X_m,X_{j'})}\in R_{x_m}} \sum_{v_m\in X_m} 1_{R_{(X_m,X_{j'})}(v_m,v_{j'})}^{\alpha_{R_{(X_m,X_{j'})}}}$. As such, for one edge type $R_{(X_m,X_{j'})}$, $\sum_{v_m\in X_m} 1_{R_{(X_m,X_{j'})}(v_m,v_{j'})}$ is the same as the edges in $E(R_{(X_m,X_{j'})})$ with $X_{j'}$ type of vertex as $v_{j'}$. This leads to the follow equality.

$$\prod_{R_{(X_m,X_{j'})} \in R_{x_m}} \sum_{v_m \in X_m} 1_{R_{(X_m,X_{j'})}(v_m,v_{j'})}^{\alpha_{R_{(X_m,X_{j'})}}} = \tag{4}$$

$$\prod_{R_{(X_m,X_{j'})} \in R_{x_m}} |E(R_{(X_m,X_{j'})})|_{X_{j'}=v_{j'}}^{\alpha_{R_{(X_m,X_{j'})}}} \tag{5}$$

[1]The extension to $\sum_{F\ni v} 1/\alpha_e \geq 1$ follows by scaling the exponents until equality holds and then applying the standard monotonicity of L^α norms on finite-measure spaces. The same trick as [24].

Algorithm 3: btedge($\mathcal{M}'$,$\mathcal{H}_{\mathcal{M}'}$)

```
1 lines 1 to 9 in Algorithm 1 but calls CrossBT(I, C');
2 Procedure CrossBT(I, C)
3     lines 11 to 12 in Algorithm 1;
4     let H' be the I and C induced subgraph of H_{M'};
      /* further filter C[A] as discussed                          */
5     foreach v ∈ C[A] do
6         foreach A' ∈ A(M') s.t. A' incident to A and appearing later than A in O do
7             if N(v, A') == ∅ in H' then C[A] ← C[A] \ {v};
8     lines 13 to 19 but calls CrossBT(I, C');
```

We progressively repeat the above steps for recursion depth $|\mathcal{A}_{\mathcal{M}'}| - 2$ till 1 (i.e., for other vertex types at the corresponding recursion depth, bearing bounds like Equation 4 in the derivation). During the repeating, we actually enforce the same R and apply the same α_R, so it can be progressively cancelled first and then becomes α_R again. As such, when exhausting $X_m \in \mathcal{A}$, we have the following inequality.

$$|Ans| \leq \prod_{R_{(X_i,X_j)} \in \mathcal{R}} |E(R_{(X_i,X_j)})|^{\alpha_{R_{(X_i,X_j)}}} \tag{6}$$

Since Finner's inequality holds for any $\alpha_{R_{(X_i,X_j)}}$ subject to the local exponent sum constraint, we can get a tight upper bound for $\mathcal{M}'$ instances by minimising $\prod_{R_{(X_i,X_j)} \in \mathcal{R}} |E(R_{(X_i,X_j)})|^{\alpha_{R_{(X_i,X_j)}}}$, which essentially solves the following linear programming (LP) optimisation problem (p-time solvable).

$$\text{minimize} \quad \sum_{R_{(X_i,X_j)} \in \mathcal{R}} \alpha_{R_{(X_i,X_j)}} log|E(R_{(X_i,X_j)})| \tag{7a}$$

$$\text{subject to} \quad \sum_{X_j \text{incident to } X_i} \alpha_{R_{(X_i,X_j)}} \geq 1 \; \forall X_i \in \mathcal{A}, \tag{7b}$$

$$\alpha_{R_{(X_i,X_j)}} \geq 0, \forall R_{(X_i,X_j)} \in \mathcal{R} \tag{7c}$$

Now, we can greatly simplify the notations, since the bound is only related to α_R for every $R \in \mathcal{R}$ and the edge list size of each edge type ($|R|$). Let $\alpha_R{}^*$ denote the optimum solution for LP 7a, the instances of $\mathcal{M}'$ can be bounded by the inequality below.

$$|Ans|^*_{edge} \leq \prod_{R \in \mathcal{R}} |R|^{\alpha_R{}^*} \tag{8}$$

We are ready to show and prove the following lemma.

Lemma 4.2. *Algorithm 3 runs in $O^*(|Ans|^*_{edge})$.*

Proof. We have analysed the bound for the total number of recursions. The cost of the reduction is absorbed by the computational cost of progressively reduced subgraph as recursion steps down, i.e., $O(|E|)$, and it is much smaller in practice. Therefore, Algorithm 3 runs in $O(|E||Ans|^*_{edge})$, that is, $O^*(|Ans|^*_{edge})$. □

Discussion. We are aware that the derived bound is similar to the AGM bound [3]. We derive the bound directly from our proposed graph algorithm using Finner's inequality, which is more concise and clear under our problem setup.

4.2 Jigsaw: filtering by bags of subgraphs

When deriving the bound and algorithm in the previous section, we treated typed edge adjacency as the fundamental filtering unit. This choice is natural, as the sizes of different edge types can be directly obtained from data. However, more informative local statistics can also be computed efficiently, such as the number of stars centred at a vertex type. Therefore, without loss of generality, we can generalise the basic building block from a single edge type to a *bag of connected edges*—that is, an edge-induced subgraph of $\mathcal{M}'$. Each such bag can serve as a certified structural unit for bounding purposes, offering the potential to derive tighter upper bounds and, consequently, more efficient algorithms.

Below, we first introduce several key concepts related to bags, and then present new upper bounds and an algorithm that takes bags as the fundamental input units.

Bag for $\mathcal{M}'$. A *bag* B of $\mathcal{M}'$ is defined as a connected subgraph of $\mathcal{M}'$ consisting of one or more edge types in $\mathcal{M}'$.

Feasible bags for $\mathcal{M}'$. Given a connected query $\mathcal{M}'$, a set of bags $\mathcal{B} = \{B_1, \ldots, B_t\}$ is said to be *feasible* for $\mathcal{M}'$ if every edge type $R \in \mathcal{R}(\mathcal{M}')$ is contained in at least one bag, i.e.,

$$\forall R \in \mathcal{R}(\mathcal{M}'),\ \exists B_j \in \mathcal{B} \text{ such that } R \subseteq B_j.$$

Such a bag set can be used to reconstruct (adhere back) the full query Q from its contained subgraphs.

Bounds with bags as input. The bound $|Ans|$ can also be bounded in terms of $|B_j|^{\alpha_j{}^*}$ for $B_j \in \mathcal{B}$ as follows ($|B_j|$ denotes the number of B_j instances), when a set of feasible bags $\mathcal{B}$ is given,

$$|Ans|^*_{bag} \leq \prod_{B_j \in \mathcal{B}} |B_j|^{\alpha_j{}^*}, \tag{9}$$

where $\alpha_j{}^*$ is the optimum solution for the following liner programming problem,

$$\text{minimize} \quad \sum_{B_j \in \mathcal{B}} \alpha_j log|B_j| \tag{10a}$$

$$\text{subject to} \quad \sum_{j:A \in B_j} \alpha_j \geq 1 \forall A \in \mathcal{A}, \tag{10b}$$

$$\alpha_j \geq 0, \quad \forall B_j \in \mathcal{B} \tag{10c}$$

The proof shares the same idea as discussed in Section 4.1, by replacing edge sets with bag sets $\mathcal{B}$, and edges incident to a vertex type with bags containing a vertex type. The critical inequality is as follows (based on Finner's inequality):

$$\sum_{v_m \in X_m} \prod_{B_j \in \mathcal{B}[X_m]} 1_{B_j} \leq \prod_{B_j \in \mathcal{B}} \sum_{v \in X_m} (1_{B_j})^{\frac{1}{\alpha_j}^{\alpha_j}}, \tag{11}$$

and it is repeatedly applied for each $X_m \in A(\mathcal{M}')$ and $\mathcal{B}[X_m]$, where $\mathcal{B}[X_m]$ denotes bags containing X_m vertex type.

Processing Q with bags. Although the generalised algorithm for processing Q shares the same idea as Algorithm 3, it is dramatically different, as it operates on subgraph instances and uses overlaps across subquery instances to determine candidate reductions. W.l.o.g., we present the generalised algorithm here and defer detailed optimisations to the following section, after introducing detailed bagging strategies.

Algorithm 4: btbag($\mathcal{H}_{\mathcal{M}'}$,$\mathcal{M}'$)

```
1  lines 1 to 3 in Algorithm 1;
2  Let B[A] be bags containg A ∀A ∈ A(M');
3  ins[B] ← instances of for each B;
4  let C[O[1]] be a vertex set of O[1] type s.t. ∀v ∈ C[O[1]], |σ_v(ins[B'])| ≥ 1
       ∀B' ∈ B[O[1]]; // σ_v(ins[B']) denotes instances of B' containing v
5  foreach v ∈ C[O[1]] do
6      I ← I ∪ {v}, ins' ← ins;
7      foreach v ∈ C[O[1]] do ins'[B] ← σ_v(ins[B]);
8      bagBT(I, ins'), remove v from I;
9  Procedure bagBT(I, ins)
10     If |I| == |A(M')| then visit I;
11     let C[O[|I| + 1]] be a vertex set of O[|I| + 1] type s.t. ∀v ∈ C[O[|I| + 1]],
         |σ_v(ins[B'])| ≥ 1 ∀B' ∈ B[O[|I| + 1]];
12     foreach v ∈ C[O[|I| + 1]] do
13         I ← I ∪ {v}, ins' ← ins;
14         foreach B containing O[|I| + 1] do
15             ins'[B] ← σ_v(ins[B]);
16         bagBT(I, ins'), remove v from I;
```

We omit the description of Algorithm 4 since it is self-explanatory and is a natural generalisation of Algorithm 3. If we omit the cost of lines 4 and 11 (which we will discuss later), Algorithm 4 runs in $O^*(|Ans|^*_{bag})$. This is because lines 4 and 11 can immediately reduce the candidate set to be the right side of Equation 11 and therefore bounded by the left side, which secures the bound in Equation 9.

Optimality discussion. So far, we have discussed optimality under a fixed set of bags. In practice, however, there exist many feasible choices of bag sets. If the number of instances of every possible bag were known in advance, we may, in principle, formulate an integer programming problem that jointly enforces feasibility and optimises the objective, with the goal of selecting an optimal bag set that yields the tightest upper bound and thus accelerates Algorithm 4. However, such an approach is unrealistic for two reasons. First, even when assuming that instance counts for all candidate bags are given, the resulting optimisation problem is NP-hard. Second, in practice, the instance counts of candidate bags are not known a priori, and storing instances incurs nontrivial overhead. In the next section, we show how to mitigate these optimisation challenges and derive tighter bounds in practice without explicitly solving the above global optimisation problem.

4.3 Strategy for creating optimal bags: stars

Previously, we analysed the overall cost of performing the search based on a feasible bag set. To further minimise this cost, two key challenges arise regardless of space consumption. First, we must accurately estimate (or correctly compute) the instances corresponding to each bag (subquery). Second, we need to minimise the total cost, as characterised by Equation 9, to achieve the most efficient execution.

For the first problem, beyond knowing the size of each edge type, we make the following observation regarding star-shaped structures centered at a vertex type.

Observation 2. *Given any vertex type $A \in \mathcal{A}_{\mathcal{M}'}$, the instance count of the star formed by all edge types incident to A can be computed in $O(|V(A)|)$ time.*

The correctness of Observation 2 follows directly when the adjacency lists of G are stored by grouping adjacent vertices according to their types.

Motivated by the above observation, we propose to construct a feasible bag set in which each bag takes a star shape, while minimising the overall search cost. We observe that enforcing the two requirements—(i) each bag corresponds to a star-shaped subgraph of $\mathcal{M}'$, and (ii) the bag set collectively covers all edge types of $\mathcal{M}'$—naturally connects the optimal bag selection problem to the weighted minimum vertex cover problem. Specifically, every vertex cover S of $\mathcal{M}'$ can be extended to a feasible star bag set as follows: for each vertex type $A \in S$, we create a bag B consisting of A and all its adjacent vertex types $N(A, \mathcal{M}')$, i.e., the subgraph of $\mathcal{M}'$ induced by $\{A\} \cup N(A, \mathcal{M}')$.

A naive approach to obtain the optimal star bag set is to enumerate every possible vertex cover, compute its corresponding $|Ans|^*_{\mathcal{B}}$ value, and then select the subset with the minimum cost. However, this strategy may miss early pruning opportunities, as the cost is evaluated only after a full vertex cover has been generated. To address this issue, we integrate cost minimisation directly with the vertex cover constraint, thereby enabling pruning during optimisation. The following linear program formalises this joint optimisation process:

$$\text{minimize} \quad \sum_{A_j \in \mathcal{A}} \alpha_j \log |B_j| \tag{12a}$$

$$\text{subject to} \quad \sum_{j: A \in B_j} \alpha_j \geq 1, \forall A \in \mathcal{A}, \tag{12b}$$

$$\alpha_j \geq 0, \quad \forall B_j \in \mathcal{B}, \tag{12c}$$

$$0 \leq \alpha_j \leq y_j, \ \forall A_j \in \mathcal{A}, \tag{12d}$$

$$y_i + y_j \geq 1, \quad \forall (A_i, A_j) \in \mathcal{R} \tag{12e}$$

Here, $y_i \in \{0, 1\}$ indicates whether the star bag B_i (centered at A_i) is included, and $\alpha_i \in [0, 1]$ denotes the exponent assigned to B_i in the cost minimisation.

Correctness. The above mixed-integer linear program (MILP) correctly identifies a set of non-zero y_i values (i.e., the selected star bags B_i) together with the optimal exponents ${\alpha_i}^*$ that minimise $\prod_{B_i \in \mathcal{B}} |B_i|^{\alpha_i}$. First, the vertex cover constraint is guaranteed by Equation 12e, where y_i and y_j are restricted to binary values $\{0, 1\}$. Second, Equations 12a–12c ensure that the chosen α_i values optimally minimise the objective in Equation 12a. Finally, Equation 12d ensures that only the selected bags (with $y_i = 1$) participate in the minimisation process.

Computational cost. Unfortunately, the problem of finding the optimal bag set under general structural constraints is NP-hard, as it subsumes the weighted minimum vertex cover problem. However, we first present a heuristic below and then show that it leads to the optimum.

Heuristic and lower bound. We propose to use the complete vertex type set—i.e., treating every vertex type in $\mathcal{M}'$ as a star centre—to provide a valid *lower bound* for the optimal cost. This is because increasing the number of bags (with great overlaps) expands the feasible region of the optimisation and exposes more opportunities for pruning, all of which can be effectively captured by the linear optimisation process of $|Ans|^*_{bag}$. In other words, the full star decomposition yields the smallest possible estimated bound under our framework, serving as a tight theoretical lower limit for any feasible bag selection.

Table 5. Datasets

Dataset	$\|\mathcal{A}\|$	$\|\mathcal{R}\|$	$\|V\|$	$\|E\|$
MovieLens [22]	5	8	2,672	209,494
DBLP [22]	5	8	37,795	349,702
Douban [22]	6	12	37,597	3,429,882
DBpedia [34]	414	1346	8,970,120	62,433,724
Freebase [34]	1,231	3,152	89,934,641	928,466,334

Why optimum. Recall that in the linear optimisation process of $|Ans|^*_{bag}$, the LP minimises the logarithmic cost, which is monotone in the set of available bags. Enlarging the set of bags (i.e., allowing more stars overlapping with existing ones) expands the feasible value of α (due to overlaps). Because the objective is minimised over this, adding bags (with overlaps) can only reduce or maintain the minimum, never increase it. Hence, using all-star bags yields the smallest achievable value.
The cost. Based on the above discussion, we can directly select all vertex types as the star centre and generate the bags. The cost of computing the bound is $O(|E|)$ plus solving the LP formulated by Equation 10a. Practically, the latter is far less than $O(|E|)$.

4.4 Last piece of the Jigsaw

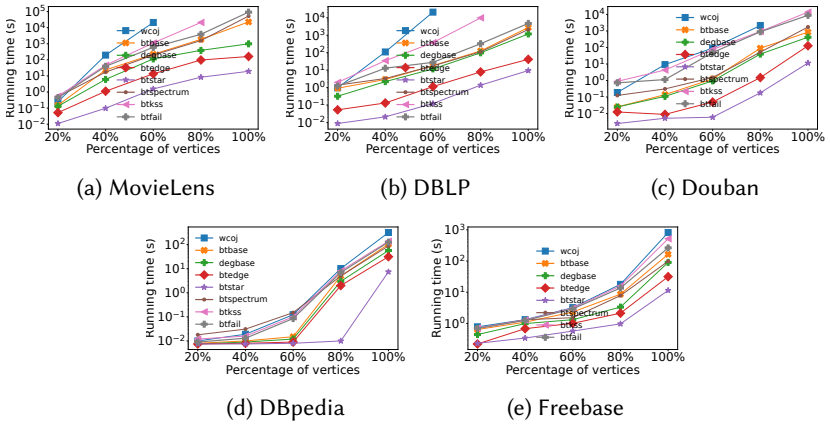
Recall that when analysing the time complexity of Algorithm 4, we omitted the cost of computing candidate sets. If this cost is not well controlled, the overall time complexity can grow explosively. In this section, we introduce a novel approach that eliminates the need to explicitly store star instances for each bag, while still allowing candidate sets in Algorithm 4 to be computed efficiently on demand.

Data structure. For each star-shaped query subgraph of $\mathcal{M}'$ contained in a bag, we store the minimal subgraph of $\mathcal{H}_{\mathcal{M}'}$ that covers all instances of that star. This minimal subgraph is represented as an adjacency list, where adjacent vertices are grouped by their types, and vertex sets are likewise organised by type.

Computing candidates (cross-star pruning). Initially, a type-A vertex v is a candidate iff, for every star bag involving A, v participates in a complete star in the corresponding minimal subgraph, which can be verified efficiently using the above data structure. W.l.o.g., once a type-A vertex v is added into the partial vertex set I, for each star bag involving A, we reduce its minimal subgraph on v by retaining only the star instances that involve v (equivalently, iteratively removing vertices/edges irrelevant to v under the star-shaped constraint). Since such conditioning is applied whenever a vertex is added to I, we call the remaining minimal subgraphs the I-jointly reduced minimal subgraphs. Based on the I-jointly reduced minimal subgraphs, to generate candidates for another type A', a type-A' vertex v' is a candidate iff, for every star bag involving A', v' participates in a complete star in the corresponding I-jointly reduced minimal subgraph. This strictly adheres to the consistency condition in Equation 11, thereby ensuring efficiency.

Space cost. Since the star-shaped structures in the bags collectively form a vertex cover, the total space cost for this part is $O(|E(\mathcal{H}_{\mathcal{M}'})|)$. Moreover, as we adopt a depth first search based traversal strategy, together with standard memory optimisations commonly used in subgraph search problems [5, 10], the working space during the search can also be maintained within $O(|E(\mathcal{H}_{\mathcal{M}'})|)$ assuming $|\mathcal{A}(\mathcal{M}')|$ is a constant.

Time complexity. The overall computation is dominated by the peeling process. Since peeling operates linearly with respect to the total size of all minimal subgraphs, the cost can be bounded



(a) MovieLens (b) DBLP (c) Douban

(d) DBpedia (e) Freebase

Fig. 4. Scalability

by $O(|E(\mathcal{H}_{\mathcal{M}'})|)$. In practice, as the minimal subgraphs are significantly reduced by vertices in I, the actual cost of computing candidates is often much lower.

Based on the above analysis, we can now formally introduce the following lemma,

Lemma 4.3. *Algorithm 4 runs in $O(|E(\mathcal{H}_{\mathcal{M}'})||Ans|^*_{star})$ (or O^* $(|Ans|^*_{star})$), where 'star' denotes star bags.*

Discussion. We are aware that there are existing indexing techniques [2, 29] that can improve the algorithm from O^* $(|Ans|^*_{star})$ to $\tilde{O}$ $(|Ans|^*_{star})$. However, since our bags are dynamically generated during the query process, building such an index incurs a non-trivial cost and would take too much space if we prebuild it for all possibilities.

Extension. It is worth noting that our algorithm applies to any $\mathcal{M}'$, as long as each edge type is treated as unique and not subject to mapping permutations. Under this assumption, edge adjacency and type consistency can be deterministically maintained throughout the search, ensuring that the same framework generalises naturally to arbitrary $\mathcal{M}'$ structures without additional modifications or symmetry-breaking procedures. The applications of such $\mathcal{M}'$ have been explored in [35].

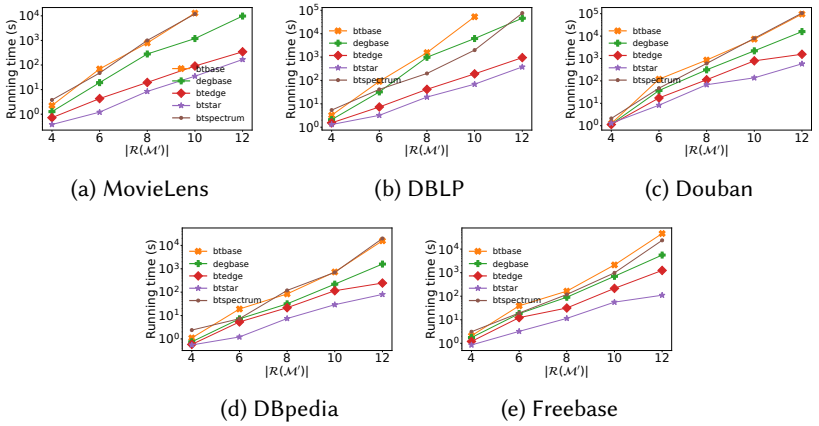
5 Experimental studies

In this section, we evaluate the effectiveness and efficiency of our proposed techniques and algorithms through extensive experiments.

Datasets. We evaluate our methods on five real-world datasets widely used in HIN research [4, 38], spanning different application domains, as shown in Table 5. The schemas of the first three HINs are small, while the last two are schema-enriched HINs.

Parameters. To systematically evaluate our proposed techniques and algorithms, we use the vertex size, diameter, and minimum degree of $\mathcal{M}'$ as parameters to group randomly generated meta-subgraphs (details discussed below). The parameter ranges are dataset dependent and are reported in the corresponding evaluation sections.

Compared algorithm. Apart from the four proposed algorithms,we also implement an online variant of the wcoj algorithm. Instead of relying on index structures, this version adopts the MergeSkip technique [8], which can be regarded as an online implementation of the Leapfrog Triejoin (LFTJ) framework, synchronously advancing iterators over multiple sorted edge lists. Moreover, we adapt SOTA techniques from SubIso to Algorithm 1 to justify our techniques for



(a) MovieLens (b) DBLP (c) Douban

(d) DBpedia (e) Freebase

Fig. 5. Varying size

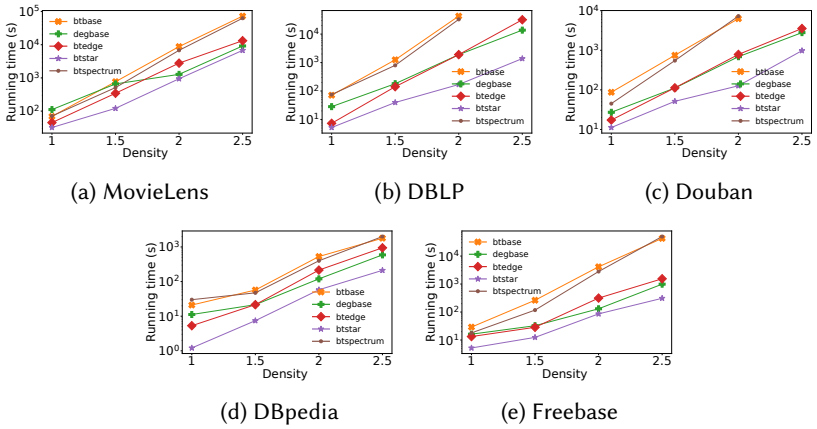
$\mathcal{M}$-instance search, yielding the following baselines. btspectrum applies a recent SubIso pruning technique [26] that filters candidates by comparing induced 2-hop neighborhoods using Laplacian spectral interlacing. btkss applies the kernel-and-shell decomposition [33] on $\mathcal{M}'$, runs Algorithm 1 on the kernel, and then extends kernel instances with shell-type vertices to obtain $\mathcal{M}$-instances. btfail applies the failure-set-based pruning using safeguard techniques [1]. Due to the difference in the problem, we can only apply an adjacency-based failure.

Queries. Borrowing ideas from previous works of subgraph searching, we generate $\mathcal{M}$ by extracting a subgraph from the schema (T) by a random walk process. For a small schema T, walking back to visited edges/vertices is allowed after all edges have been visited (extended $\mathcal{M}'$). For small schema HINs, walking back to visited edges/vertices is allowed after all edges have been visited. Each pool contains 20 queries for the first three datasets due to their small network schemas. For DBpedia and Freebase, the pool sizes are increased to 50 and 80, respectively, to enable more comprehensive evaluation given their larger schemas. We note that subgraph matching studies typically use large query pools, which are less applicable in our setting since network schemas are substantially smaller than the underlying network data.

Measures. We measure the algorithm's runtime as total CPU time (in seconds), excluding I/O costs for loading the graph from disk to main memory. A 4-hour timeout is set. Experiments are conducted on a Mac with an M4 Max CPU, 128GB of memory, and macOS 26.2. Each algorithm runs no less than 10 times if the running time is less than 2 hours (3 times otherwise). The average results are reported.

Scalability against SOTAs. For each dataset, we select fixed percentages of vertices of each vertex type and compute their induced subgraphs. For each fixed percentage other than 100% in Figure 4, 10 different sets of $\mathcal{V}$ are randomly selected on which algorithms work. Figure 4 reports the scalability results. Based on their performance, the evaluated algorithms can be categorised into three groups: $A = \{\text{wcoj}, \text{btkss}, \text{btfail}\}$, $B = \{\text{btspectrum}, \text{btbase}, \text{degbase}\}$, and $C = \{\text{btedge}, \text{btstar}\}$. Algorithms in group A are consistently slower than those in group B, while group B is consistently slower than group C across all datasets, with the performance gaps being pronounced on the more challenging MovieLens and DBLP datasets.

The poor practical performance of wcoj is mainly due to its need to progressively sort typed edge lists on-the-fly according to the search order in order to efficiently perform cross-merging and preserve its worst-case time complexity, which introduces substantial overhead in practice.



(a) MovieLens (b) DBLP (c) Douban

(d) DBpedia (e) Freebase

Fig. 6. Varying $\mathcal{M}'$ density

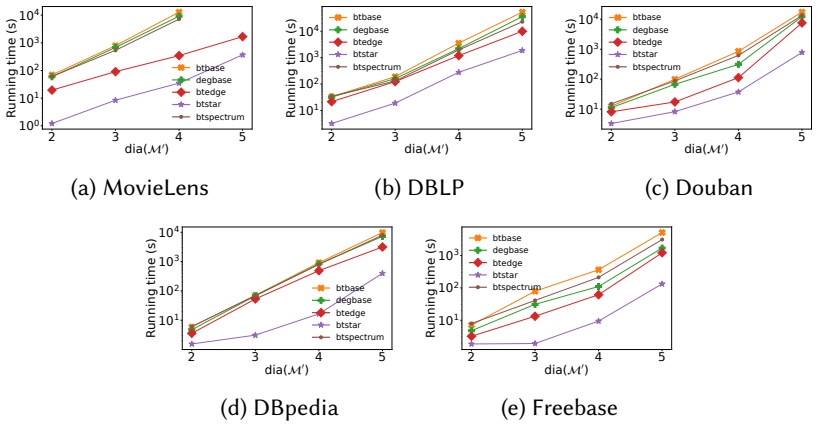
For the three baselines with SOTA techniques for SubIso, btspectrum is faster than btkss and btfail. btkss is not well-suited to our problem setting. It decomposes a query into vertex-disjoint components (kernel and shells), thereby breaking global structural dependencies and limiting pre-pruning under type-aware adjacency constraints. This design targets streaming SubIso scenarios with dynamic edge updates, whereas our problem is static and dominated by joint adjacency interactions. As a result, the decomposition provides limited pruning benefit, while AGM-bag-based decompositions preserve global structure and yield tighter worst-case bounds and better practical performance.

btfail performs poorly in our setting due to a fundamental mismatch in failure semantics. In SubIso, guardmask-based pruning is effective because many failures arise from injective mapping propagation, adjacency violations under partial embeddings, and their cascading effects, where a failure set is a minimal subset of a partial result causing the failure. However, our problem does not enforce injectivity, and failures arise solely from the type-aware adjacency constraints under partial results, whose failure sets often cannot be detected by guardmask conditions. As a result, btfail incurs non-trivial overhead while providing limited pruning power.

btspectrum achieves stronger pruning than btfail with slightly lower overhead. However, its effectiveness is inherently limited in our setting because spectral decomposition ignores vertex and edge types. Consequently, it cannot distinguish vertices that are structurally similar in an untyped sense but incompatible under type-aware adjacency constraints. Besides, the pruning overhead is nontrivial. Compared to our type-aware cross-adjacency-based pruning, btspectrum often retains candidates whose untyped structure matches the pattern $\mathcal{M}'$, and thus cannot prune them effectively. As a result, although btspectrum may outperform our weaker baselines in some cases, it consistently exhibits a performance gap relative to btedge and btstar.

Since wocj, btkss, and btfailure consistently perform slowly, we exclude them from the experiments below to better demonstrate the differences over other methods.

Varying $\mathcal{M}'$ size. Figure 5 shows the running time when varying $|\mathcal{A}(\mathcal{M}')|$ for each algorithm on every dataset. As expected, all algorithms spend more time as $|\mathcal{A}(\mathcal{M}')|$ increases. This is mainly because all algorithms try to type vertex sets. btbase performs poorly because it lacks effective pruning to reduce time complexity, even though it explores neighbour-based reductions within its search space. degbase is slightly better than btbase because the forward-neighbours-based reduction can further improve neighbour-based reductions. btspectrum performs slightly faster than btbase



(a) MovieLens (b) DBLP (c) Douban

(d) DBpedia (e) Freebase

Fig. 7. Varying $\mathcal{M}'$ diameter

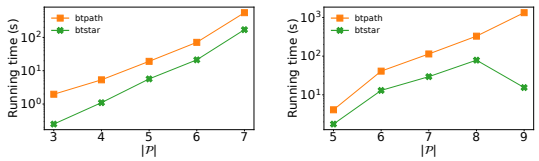
in most datasets because it is capable of pruning more candidates. However, the pruning overhead weakens the pruning effectiveness. For btedge and btstar, the two algorithms with non-trivial time complexities, perform much better compared to baselines for all datasets. This is because they are equipped with more solid pruning ideas. Star cover-based pruning clearly offers greater benefits with no increased time complexity, resulting in the performance gap between btedge and btstar.

Varying density of $\mathcal{M}'$. Figure 6 shows the running time as we vary the density for $\mathcal{M}'$. The density is measured by $\frac{|\mathcal{R}(\mathcal{M}')|}{|\mathcal{A}(\mathcal{M}')|}$ while $|\mathcal{A}(\mathcal{M}')|$ no larger than 6. The overall trends are similar to previous experiments, but one difference is noticeable, i.e., degbase can beat btedge at high density. By carefully checking the queries in the pool, we find that when density is high, the query subgraph tends to have short diameters, i.e. 1 or 2. This explains why degbase performs well since degeneracy order can well bound 1-hop neighbours, and if all different types of neighbours are just 1-hop neighbours, the time complexity of degbase can be improved in terms of δ, which is even tighter than $|Ans|^*_{edge}$. On the other hand, when the diameters are small, stars in the star cover exhibit high overlap, which, in fact, enhances pruning effectiveness since candidates must involve all star instances. In fact, such pruning can lead to the candidate set containing vertices in the final result only. This makes btstar still faster than degbase.

Varying diameter of $\mathcal{M}$. The above experiments immediately prompt us to examine how algorithms perform as we vary the diameters. Figure 7 shows the result. In this case, degbase loses its advantages. In contrast, btstar performs much better as the diameter increases. This is because star-based cover can observe dependencies over vertices much better than the neighbouring-based method. The longer the diameter, the lower the pruning effectiveness of the neighbour-based method. By controlling pruning costs effectively, btstar performs well. This experiment also indicates that if we can well control the pruning cost, using longer (larger) subgraphs of $\mathcal{M}'$ with overlap can perform even better.

Evaluations on meta-path. We compare our best algorithm, btstar, with meta-path instance search algorithms adapted from [32, 37], denoted as btpath. Given a meta-path $\mathcal{P}$, btpath can be viewed as a simplified version of btbase, which enumerates combinations by following required neighbour types along $\mathcal{P}$. Figure 8 reports the performance while varying the path length $|\mathcal{P}|$. On Douban, the running times of both methods increase with $|\mathcal{P}|$, as the schema is compact and longer paths induce more $\mathcal{P}$ instances. In contrast, on Freebase, the running time of btstar decreases

when $|\mathcal{P}| > 8$. This is because Freebase is schema-enriched, and longer paths tend to traverse weakly connected parts of HINs, resulting in fewer $\mathcal{P}$ instances. This trend is captured by btstar, as its running time is bounded by the AGM star bound. Across both datasets, btstar consistently outperforms btpath due to prefiltering and cross-star-bag–based pruning.



(a) Douban (b) Freebase

Fig. 8. $\mathcal{P}$ instance search

Case study. We demonstrate the effectiveness of $\mathcal{M}$ instances for discovering cohesive subgraphs on the DBLP dataset. We adopt the density model from [4] and employ $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{M}_1$ shown in Table 1. For the dense subgraphs discovered under these patterns, denoted by $H_{\mathcal{P}_1}$, $H_{\mathcal{P}_2}$, and $H_{\mathcal{M}_1}$, we report the average pairwise vertex similarities measured by PathSim [28] and HeteSim [25] w.r.t. the corresponding meta-paths. PathSim measures the similarity between pairs of vertices of the same type with respect to a symmetric meta-path, while HeteSim measures the similarity between pairs of vertices of different types with respect to an asymmetric meta-path. For example, a score of 0.72 indicates the average PathSim value between pairs of A-type vertices along $\mathcal{P}_1$ within $H_{\mathcal{P}_1}$. As discussed in [4], since one of the purposes of a dense subgraph is to find high similarity vertices, the higher the score, the better the result. Notably, vertices in $H_{\mathcal{M}_1}$ consistently exhibit simultaneous high similarity under both measures, revealing cohesive structures that cannot be identified using a meta-path alone.

6 Related works

Meta-structure. The idea of extending $\mathcal{P}$ to $\mathcal{M}$ is proposed in [17], which considers a single source and target $\mathcal{M}$ that is layered and acyclic. A backtracking algorithm is proposed that progressively explores candidate subgraphs of acyclic and layered $\mathcal{M}$ to derive instances of $\mathcal{M}$. Additionally, an index is proposed to accelerate counting. [36] and [39] present matrix multiplication based method for counting $\mathcal{M}$ instances. $\mathcal{M}$ studied in this work is more general; the proposed algorithm can deal with any $\mathcal{M}$ induced by subset edges of T. Moreover, recent studies such as [6] explore how to generate high-quality $\mathcal{M}$ for downstream applications. Our work is complementary to [6] and focuses on efficiently searching $\mathcal{M}$-instances given an input $\mathcal{M}$.

Worst case optimal join (WCOJ). We establish nontrivial bounds for our algorithms and $\mathcal{M}$-instances within the worst-case optimal join (WCOJ), i.e., AGM-bound, framework [3]. Our derivation adopts a graph-structural view via Finner's inequality, whereas existing WCOJ analyses are typically developed for conjunctive queries through entropy-based arguments and Hölder-type inequalities [7, 23, 24]. Thus, our analysis can be seen as a problem-specific instance of the WCOJ framework with a concise proof tailored to graphs. Moreover, unlike conventional WCOJ algorithms over relational tables, our algorithms follow a subgraph-search paradigm that recursively refines candidate sets and backtracks to enumerate instances while matching the AGM bound.

Subgraph search in simple or labelled graphs. Subgraph matching has been extensively studied. The most popular approaches lay in faltering-ordering-enumerating framework [15, 20, 27] and decomposition-plus-join framework [21]. As discussed, our studied problem is essentially different

and simpler than the subgraph search. Our problem does not incur the cost of attempting mapping (embedding), as it exhibits a permutation search space. In contrast, our problem hardness is bounded by the structure exploration cost (the sum of vertex combinations). Popular SubIso pruning offers limited benefit in our setting. Equivalence-based symmetry breaking (e.g., NEC) [15] requires interchangeable query roles, but edge types in $\mathcal{M}$ and vertex types in $\mathcal{M}'$ are unique, making vertices role-distinct with little symmetry to exploit. Failure-set/guard-based pruning [1] mainly targets injective and local-adjacency failures; without injectivity, our failures are higher-order and type-aware, so local guards often cannot find minimal failure sets and thus provide limited reuse. Spectral pruning [26] is based on an untyped view and thus misses vertex/edge types, retaining untyped-similar yet type-incompatible candidates. Therefore, our pruning approaches differ from SubIso and rely on type-aware, structure-driven pruning.

Cohesive subgraph search over HINs. Many novel cohesive models have been proposed for HINs based on well-studied cohesive models in unipartite and bipartite graphs. For instance, k-core model has been adapted to $(k, \mathcal{P})$-core [11, 19] and relational community [18], k-truss to $(k, \mathcal{P})$-truss [34], and k-clique to m-clique [16]. The butterfly-core model [9] combines both the k-core and k-bitruss models. Most of them aim to group the same type of vertices cohesively for different application scenarios. Besides, there are extensions of $(k, \mathcal{P})$-core for discovering influential communities [37] and structurally similar communities [32] in HINs. A few of them [18, 30, 31] can be adapted to search cohesive multipartite subgraphs. Our work is orthogonal to these works and can provide building blocks for these models. For instance, we can generalize $(k, \mathcal{P})$-core to $(k, \mathcal{M})$-core. There are also studies such as [6] that explore how to generate high-quality $\mathcal{M}$ for downstream applications.

7 Conclusion

In this paper, we generalise the concept of meta-path $\mathcal{P}$ to meta-subgraph $\mathcal{M}$ and design efficient algorithms for enumerating all its instances. We show that $\mathcal{M}$, expressed in an edge-type adjacency list, can be transformed into $\mathcal{M}'$, represented by an annotated vertex-type adjacency list, enabling instance enumeration through vertex combinations. Based on $\mathcal{M}'$, we propose effective pre-pruning techniques that substantially reduce G. A nontrivial backtracking algorithm is then developed, which prunes candidate sets using both edge-type adjacency in $\mathcal{M}'$ and data adjacency in partial results. We prove that it runs in $O(|E||Ans|^*_{edge})$ from a graph-structural perspective. We further extend pruning from edge-type to bag adjacency, where each bag is a subgraph of $\mathcal{M}'$ covering all its edges. In particular, star-bag decomposition yields tighter bounds, and star instances can be efficiently derived without materialisation. Combining all techniques, we obtain an improved algorithm with guaranteed $O(|E||Ans|^*_{star})$ complexity, which is practically faster. Extensive experiments validate the efficiency and effectiveness of our approach.

Acknowledgments

This work is jointly supported by the ARC Discovery Projects under Grant Nos. DP220102191, DP240101591 and DP250100536, the ARC DECRA under Grant No. DE240100200, and the National Natural Science Foundation of China under Grant Nos. 62572336 and 62272334.

References

[1] Junya Arai, Yasuhiro Fujiwara, and Makoto Onizuka. 2023. GuP: Fast Subgraph Matching by Guard-based Pruning. *Proc. ACM Manag. Data* 1, 2, Article 167 (June 2023), 26 pages. doi:10.1145/3589312

[2] Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, Juan L. Reutter, Javiel Rojas-Ledesma, and Adrián Soto. 2021. Worst-Case Optimal Graph Joins in Almost No Space. In *Proceedings of the 2021 International Conference on Management*

of Data (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 102–114. doi:10.1145/3448016.3457256

[3] Albert Atserias, Martin Grohe, and Dániel Marx. 2008. Size Bounds and Query Plans for Relational Joins. In *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*. IEEE, 739–748. doi:10.1109/FOCS.2008.34

[4] Lu Chen, Chengfei Liu, Rui Zhou, Kewen Liao, Jiajie Xu, and Jianxin Li. 2023. Densest Multipartite Subgraph Search in Heterogeneous Information Networks. *Proc. VLDB Endow.* 17, 4 (Dec. 2023), 699–711. doi:10.14778/3636218.3636226

[5] Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, and Jianxin Li. 2022. Efficient maximal biclique enumeration for large sparse bipartite graphs. *Proc. VLDB Endow.* 15, 8 (April 2022), 1559–1571. doi:10.14778/3529337.3529341

[6] Lin Chen, Fengli Xu, Nian Li, Zhenyu Han, Meng Wang, Yong Li, and Pan Hui. 2024. Large Language Model-driven Meta-structure Discovery in Heterogeneous Information Network. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 307–318. doi:10.1145/3637528.3671965

[7] F.R.K Chung, R.L Graham, P Frankl, and J.B Shearer. 1986. Some intersection theorems for ordered sets and graphs. *Journal of Combinatorial Theory, Series A* 43, 1 (1986), 23–37. doi:10.1016/0097-3165(86)90019-1

[8] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. 2000. Adaptive set intersections, unions, and differences. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, California, USA) *(SODA '00)*. Society for Industrial and Applied Mathematics, USA, 743–752.

[9] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-Core Community Search over Labeled Graphs. *Proc. VLDB Endow.* 14, 11 (jul 2021), 2006–2018. doi:10.14778/3476249.3476258

[10] David Eppstein and Darren Strash. 2011. Listing All Maximal Cliques in Large Sparse Real-World Graphs. In *Experimental Algorithms*, Panos M. Pardalos and Steffen Rebennack (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 364–375.

[11] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and Efficient Community Search over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 13, 6 (feb 2020), 854–867. doi:10.14778/3380750.3380756

[12] Helmut Finner. 1992. A Generalization of Holder's Inequality and Some Probability Inequalities. *The Annals of Probability* 20, 4 (1992), 1893 – 1901. doi:10.1214/aop/1176989534

[13] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (Singapore, Singapore) *(CIKM '17)*. Association for Computing Machinery, New York, NY, USA, 1797–1806. doi:10.1145/3132847.3132953

[14] Yucan Guo, Chenhao Ma, and Yixiang Fang. 2024. Efficient Core Decomposition Over Large Heterogeneous Information Networks. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 2393–2406. doi:10.1109/ICDE60146.2024.00189

[15] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. 2013. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, New York, USA) *(SIGMOD '13)*. Association for Computing Machinery, New York, NY, USA, 337–348. doi:10.1145/2463676.2465300

[16] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian Y.H. Lam. 2019. Discovering Maximal Motif Cliques in Large Heterogeneous Information Networks. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 746–757. doi:10.1109/ICDE.2019.00072

[17] Zhipeng Huang, Yudian Zheng, Reynold Cheng, Yizhou Sun, Nikos Mamoulis, and Xiang Li. 2016. Meta Structure: Computing Relevance in Large Heterogeneous Information Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1595–1604. doi:10.1145/2939672.2939815

[18] Xun Jian, Yue Wang, and Lei Chen. 2021. Effective and Efficient Relational Community Detection and Search in Large Dynamic Heterogeneous Information Networks. *Proc. VLDB Endow.* 13, 10 (mar 2021), 1723–1736. doi:10.14778/3401960.3401969

[19] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. 2022. Effective Community Search over Large Star-Schema Heterogeneous Information Networks. *Proc. VLDB Endow.* 15, 11 (sep 2022), 2307–2320. doi:10.14778/3551793.3551795

[20] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong, and Wook-Shin Han. 2022. Fast subgraph query processing and subgraph matching via static and dynamic equivalences. *The VLDB Journal* 32, 2 (June 2022), 343–368. doi:10.1007/s00778-022-00749-x

[21] Qiyan Li, Jeffrey Xu Yu, and Zongyan He. 2025. Subgraph Matching: A New Decomposition Based Approach. *Proc. VLDB Endow.* 18, 11 (Sept. 2025), 4282–4294. doi:10.14778/3749646.3749693

[22] librahu. 2019. HIN-Datasets-for-Recommendation-and-Network-Embedding. https://github.com/librahu/HIN-Datasets-for-Recommendation-and-Network-Embedding

[23] Hung Q. Ngo. 2018. Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Houston, TX, USA) *(PODS '18)*. Association for Computing Machinery, New York, NY, USA, 111–124. doi:10.1145/3196959.3196990

[24] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case Optimal Join Algorithms. *J. ACM* 65, 3, Article 16 (March 2018), 40 pages. doi:10.1145/3180143

[25] Chuan Shi, Xiangnan Kong, Yue Huang, S Yu Philip, and Bin Wu. 2014. Hetesim: A general framework for relevance measure in heterogeneous networks. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2014), 2479–2492.

[26] Konstantinos Skitsas, Davide Mottin, and Panagiotis Karras. 2025. Pilos: Scalable Large-Subgraph Matching by Online Spectral Filtering. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. 1180–1193. doi:10.1109/ICDE65448.2025.00093

[27] Shixuan Sun, Xibo Sun, Yulin Che, Qiong Luo, and Bingsheng He. 2020. RapidMatch: a holistic approach to subgraph query processing. *Proc. VLDB Endow.* 14, 2 (Oct. 2020), 176–188. doi:10.14778/3425879.3425888

[28] 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 (Aug. 2011), 992–1003. doi:10.14778/3402707.3402736

[29] Todd L. Veldhuizen. 2012. Leapfrog Triejoin: a worst-case optimal join algorithm. *ArXiv* abs/1210.0481 (2012). https://api.semanticscholar.org/CorpusID:26626787

[30] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex Priority Based Butterfly Counting for Large-Scale Bipartite Networks. *Proc. VLDB Endow.* 12, 10 (jun 2019), 1139–1152. doi:10.14778/3339490.3339497

[31] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient Bitruss Decomposition for Large-scale Bipartite Graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 661–672. doi:10.1109/ICDE48307.2020.00063

[32] Shu Wang, Yixiang Fang, and Wensheng Luo. 2025. Searching and Detecting Structurally Similar Communities in Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 18, 5 (Jan. 2025), 1425–1438. doi:10.14778/3718057.3718070

[33] Rongjian Yang, Zhijie Zhang, Weiguo Zheng, and Jeffrey Xu Yu. 2023. Fast Continuous Subgraph Matching over Streaming Graphs via Backtracking Reduction. *Proc. ACM Manag. Data* 1, 1, Article 15 (May 2023), 26 pages. doi:10.1145/3588695

[34] Yixing Yang, Yixiang Fang, Xuemin Lin, Wenjie Zhang, and Yixiang Fang. 2020. Effective and Efficient Truss Computation over Large Heterogeneous Information Networks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 901–912. doi:10.1109/ICDE48307.2020.00083

[35] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-Graph Based Recommendation Fusion over Heterogeneous Information Networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 635–644. doi:10.1145/3097983.3098063

[36] Yuyan Zheng, Jianhua Qu, and Jiajia Yang. 2024. StructSim: Meta-Structure-Based Similarity Measure in Heterogeneous Information Networks. *Applied Sciences* 14, 2 (2024). doi:10.3390/app14020935

[37] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential Community Search over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 16, 8 (April 2023), 2047–2060. doi:10.14778/3594512.3594532

[38] Yingli Zhou, Yixiang Fang, Chenhao Ma, Tianci Hou, and Xin Huang. 2024. Efficient Maximal Motif-Clique Enumeration over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 17, 11 (July 2024), 2946–2959. doi:10.14778/3681954.3681975

[39] Yu Zhou, Jianbin Huang, Heli Sun, Yizhou Sun, Shaojie Qiao, and Stephen Wambura. 2019. Recurrent Meta-Structure for Robust Similarity Measure in Heterogeneous Information Networks. *ACM Trans. Knowl. Discov. Data* 13, 6, Article 64 (Dec. 2019), 33 pages. doi:10.1145/3364226

Received October 2025; revised January 2026; accepted February 2026