A Comprehensive Survey and Experimental Study of Subgraph Matching: Trends, Unbiasedness, and Interaction

ZHIJIE ZHANG, School of Data Science, Fudan University, China
YUJIE LU, School of Data Science, Fudan University, China
WEIGUO ZHENG*, School of Data Science, Fudan University, China
XUEMIN LIN, Antai College of Economics and Management, Shanghai JiaoTong University, China

Subgraph matching is a fundamental problem in graph analysis. In recent years, many subgraph matching algorithms have been proposed, making it pressing and challenging to compare their performance and identify their strengths and weaknesses. We observe that (1) The embedding enumeration in the classic filtering-ordering-enumerating framework dominates the overall performance, and thus enhancing the backtracking paradigm is becoming a current research trend; (2) Simply changing the limitation of output size results in a substantial variation in the ranking of different methods, leading to biased performance evaluation; (3) The techniques employed at different stages of subgraph matching interact with each other, making it less feasible to replace and evaluate a single technique in isolation.

Therefore, a comprehensive survey and experimental study of subgraph matching is necessary to identify the current trends, ensure unbiasedness, and investigate the potential interactions. In this paper, we comprehensively review the methods in the current trend and experimentally confirm their advantage over prior approaches. We unbiasedly evaluate the performance of these algorithms by using an effective metric, namely embeddings per second. To fully investigate the interactions between various techniques, we select 10 representative techniques for each stage and evaluate all the feasible combinations.

CCS Concepts: • **Information systems** → **Information retrieval query processing**; **Graph-based database models**.

Additional Key Words and Phrases: Subgraph Matching, Graph Mining, Experimental Study

ACM Reference Format:
Zhijie Zhang, Yujie Lu, Weiguo Zheng, and Xuemin Lin. 2024. A Comprehensive Survey and Experimental Study of Subgraph Matching: Trends, Unbiasedness, and Interaction. *Proc. ACM Manag. Data* 2, 1 (SIGMOD), Article 60 (February 2024), 29 pages. https://doi.org/10.1145/3639315

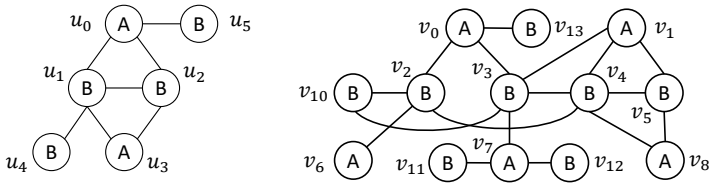
1 INTRODUCTION

The search for a particular structure within a large graph is a fundamental operation in graph mining, formally defined as subgraph matching. As shown in Figure 1, given a large data graph G and a query graph Q, subgraph matching enumerates all embeddings f, which map every vertex in Q to the corresponding vertex of an isomorphic subgraph in G.

*Corresponding Author: zhengweiguo@fudan.edu.cn

Authors' addresses: Zhijie Zhang, School of Data Science, Fudan University, Shanghai, China, zhangzj22@m.fudan.edu.cn; Yujie Lu, School of Data Science, Fudan University, Shanghai, China, yjlu23@m.fudan.edu.cn; Weiguo Zheng, School of Data Science, Fudan University, Shanghai, China, zhengweiguo@fudan.edu.cn; Xuemin Lin, Antai College of Economics and Management, Shanghai JiaoTong University, Shanghai, China, xuemin.lin@gmail.com.

ACM 2836-6573/2024/2-ART60
https://doi.org/10.1145/3639315



(a) Query Graph Q (b) Data Graph G

Fig. 1. A running example of Subgraph Matching.

1.1 Background

Subgraph matching finds broad applications in various fields such as fraud detection [62], social network analysis [19], bioinformatics [9], and knowledge graphs [40, 60]. Subgraph matching is a typical NP-hard problem with the worst-case time complexity $O(|V_G|^{|V_Q|})$, where $|V_G|$ and $|V_Q|$ denote the number of vertices in graphs G and Q, respectively [22]. Although the size of the query graph is often bounded in practical application scenarios, subgraph queries with tens of nodes are widely studied in bioinformatics [9], social network [49], and knowledge graphs [48]. Therefore, the computational overhead for queries of this size remains prohibitively expensive and it may become a bottleneck in the applications. Numerous efficient algorithms [6, 7, 11, 26, 28, 34, 38, 65, 66, 75, 81] have been proposed for subgraph matching. In addition, there are studies exploring alternative settings, including parallel [32, 42, 84] and distributed subgraph matching [93, 98], continuous subgraph matching [41, 55, 76, 95], as well as the GPU-based [24, 77, 88, 97] or FPGA-based approaches [33, 89].

In this paper, we focus on subgraph matching within a single large graph under the basic setting, i.e., in-memory, CPU-based, and single-threaded approaches. This forms the basis for studying subgraph matching, enabling its extension to diverse scenarios. Given the crucial role of subgraph matching, a comprehensive survey is warranted that goes beyond merely enumerating existing algorithms, to provide an unbiased evaluation of techniques and insights into future research.

The objective of our survey is to address three central questions in the domain of subgraph matching.

(1) What are the prevailing trends in algorithm development?
(2) How to rigorously assess the performance of an algorithm without any bias?
(3) How does the interaction between techniques affect the performance evaluation?

1.2 Contributions

Trends. Most methods for subgraph matching follow the *filtering-ordering-enumerating* framework. In the last decade, the main focus of existing algorithms has been on filtering and ordering optimization [6, 7, 9, 26, 28, 35, 65]. Despite equipment with carefully designed candidate filtering and ordering, backtracking algorithms still face a significant issue of numerous futile recursions. Recently, a growing effort has been made to enhance the backtracking framework to minimize the number of backtrackings. This has led to the development of several noteworthy works in this field, such as DPiso [25], RM [75], VEQ [38], GuP [4], and CaLiG [95]. All these methods involve modifying the backtracking framework to varying degrees, which has become a current research trend. Thus, we believe that a comprehensive review is needed to summarize and further investigate these works in greater depth.

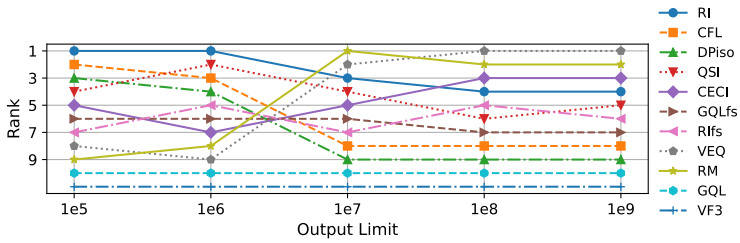


Fig. 2. Ranking changing vs. output limit.

Reducing backtrackings could be primarily achieved via two means: pruning during enumeration to avoid unnecessary and redundant backtrackings (DPiso, VEQ, and GuP); and enumerating multiple embeddings simultaneously (RM and CaLiG). *We offer a comprehensive comparative analysis of these algorithms and investigate their individual strengths and weaknesses through empirical evaluation.*

Unbiasedness. We notice that previous works [4, 25, 38, 74] follow a "conventional setting" in evaluations: an algorithm is terminated either after producing 10^5 matchings or upon reaching a timeout of 300 seconds. In the early stage of subgraph matching studies, a size of 10^5 matchings may have been considered representative. However, with the increasing scale of query and data graphs, continuing to follow this setting might be inappropriate. We evaluate 9 algorithms on dataset *dblp* and report their rankings under different output limits (i.e., the maximum number of embeddings) in Figure 2. It can be observed that simply adjusting the output limit results in a dramatic shift in the rankings. As the number of reported embeddings increases, the previously high-ranked methods RI [9] and CFL [7] start to decline, while initially low-ranked methods such as RM [75] and VEQ [38] rise to the forefront. Additionally, as shown in Figure 12 (Section 4.5), the results for all extended combinations exhibit even more significant changes.

This phenomenon is believed to sound the alarm for researchers in the field of subgraph matching, raising the critical questions: *How can we unbiasedly evaluate the performance of an algorithm?* The ideal solution is to conduct experiments without any limits, however, it is not realistic due to the NP-hard nature of subgraph matching [22]. In our paper, we introduce the metric *embeddings per seconds* (EPS), which is defined as the average number of embeddings returned per second. This metric provides a comprehensive measurement by considering both time cost and the number of reported embeddings. EPS allows for comparisons across different time and output limits, ensuring relative consistency in performance, as illustrated in Figure 13. We also re-evaluate representative subgraph matching algorithms using the metric EPS.

Interaction. The abundant algorithms for subgraph matching could be decomposed into individual techniques (filter, order, and enumeration) and be combined with each other, allowing a large investigation space of the effect of these techniques. However, previous surveys [47, 74] either used the original algorithms or replaced individual techniques based on the default algorithm, overlooking the potential interactions among diverse algorithm combinations. In fact, for each individual technique, different complementary techniques may be required to achieve its optimal performance. Assume that there are a filtering, b ordering, and c enumerating techniques, the combination space covers $a \times b \times c$ possible combinations, but in the previous studies [74], only $a + b + c$ of combinations have been investigated, resulting in a significant number of combinations being overlooked and left unexplored.

To thoroughly investigate the entire design space, this paper selects 10 representative techniques for each stage and explores the feasible combinations. Through a comprehensive analysis, we

evaluate the performance of both the original combinations and the feasible combinations derived from various algorithms. The presence of interaction effects, as revealed by our experimental findings, holds greater significance. Specifically, *our experiments demonstrate that several enumeration methods are significantly influenced by the effectiveness of filters and matching orders, highlighting the importance of considering these factors in performance evaluation.*

In summary, we make the following contributions in this paper.

- We identify a notable shift in the field of subgraph matching, where the focus is transitioning from filtering and ordering optimizations to the development of robust backtracking techniques.
- We have discovered the potential bias caused by using the conventional setting that limits the reported embeddings. Thus, we introduce an effective metric, namely embeddings per second (EPS), which reflects the performance of algorithms consistently.
- Beyond the original methods, we use 10 typical techniques for each stage and evaluate all the feasible combinations to comprehensively investigate the interactions between these techniques.
- We conduct extensive experiments on various real-world datasets, as well as diverse types of synthetic datasets and label assignments, which provides a thorough analysis of the performance of all technique combinations.

2 PRELIMINARY

2.1 Problem Definition

In this paper, we focus on undirected and connected graphs with labeled vertices. Note that graphs with labeled edges will also be discussed in Section 4.3. Generally, we consider query graph $Q = (V_Q, E_Q, \Sigma, L_Q)$ and data graph $G = (V_G, E_G, \Sigma, L_G)$, where V_Q and V_G are sets of vertices, E_Q and E_G are sets of edges, Σ is a set of labels, and $L_Q : V_Q \rightarrow \Sigma$ and $L_G : V_G \rightarrow \Sigma$ are the mappings of a query (resp. data) vertex to its label. If there is no ambiguity, we use u as a query vertex and v as a data vertex. Let $d(u)$ denote the degree of vertex u and $N_Q(u)$ (resp. $N_G(v)$) denote the sets of neighbors of u (resp. v).

Definition 2.1 (Subgraph Isomorphism). *Given a query graph $Q = (V_Q, E_Q, \Sigma, L_Q)$ and a data graph $G = (V_G, E_G, \Sigma, L_G)$, Q is subgraph isomorphic to G if there exists a mapping function $f: V_Q \rightarrow V_G$, such that*

(1) $\forall\, u \in V_Q$, we have $L_Q(u) = L_G(f(u))$ where $f(u) \in V_G$,
(2) $\forall\, e(u_1, u_2) \in E_Q$, we have $e(f(u_1), f(u_2)) \in E_G$, and
(3) $\forall\, u_i, u_j \in E_Q, u_i \neq u_j$), then $f(u_i) \neq f(u_j)$.

The mapping function f is also called an *embedding* of Q in G. Each embedding could be expressed as a set of one-to-one vertex pairs $\{(u, f(u))\}$, and each vertex pair is called an assignment.

A partial embedding $M : I \rightarrow V_G$, where $I \subsetneq V_Q$, is an embedding of a subgraph of Q (induced by I). An extension is to add an assignment (u, v) to a partial embedding M, denoted by $M \cup (u, v)$.

Definition 2.2 (Subgraph Matching). *Given a query graph Q and data graph G, subgraph matching returns all the embeddings of Q in G.*

Example 2.1.1. Considering the query Q and data graph G in Figure 1, according to the definition of subgraph isomorphism (Definition 2.1), $M = \{(u_0, v_1), (u_1, v_4), (u_2, v_5), (u_3, v_8), (u_4, v_2), (u_5, v_3)\}$ is an embedding of Q in G.

Definition 2.2 returns all embeddings, implying automorphisms can yield multiple solutions that may be unnecessary computation in some applications. A stream of researches [29, 64, 68]

Algorithm 1: General Framework for Subgraph Matching

```
   Input: a query graph Q and a data graph G
   Output: All the embedding of Q into G
1  C ← FilterVertices(Q, G)
2  φ ← GenerateOrder(Q, G, C)
3  Enumerate(Q, G, C, φ, {}, 0)
4  Function Enumerate(Q, G, C, φ, M, i):
5      if Termination condition met then
6          output TerminateFunc(Q, G, C, φ, M)
7      u ← SelectNextVertex(Q, G, C, φ, M)
8      C_M(u) ← ComputeLocalCandidates(Q, G, C, φ, M, i)
9      foreach v ∈ C_M(u) do
10         if Pruning condition met then
11             PruningFunc(Q, G, C, φ, M)
12         else
13             Extend M by (u, v)
14             Enumerate(Q, G, C, φ, M, i + 1)
15             Update the index for pruning
16             Delete (u, v) from M
```

only delivers induced subgraphs, i.e., automorphism producing one single embedding at most. For empirical studies, please refer to Section 4.4.

The general framework of filtering-ordering-enumerating is outlined in Algorithm 1.

Filtering Vertices. The algorithm first generates candidate sets by filtering vertices (line 1). Compared to directly traversing the entire graph, the filtering method selects candidates for each query vertices and eliminates data vertices that cannot be potential matches. The candidate of query vertex u is denoted by $C(u)$.

Matching Order. The algorithm then generates a matching order (line 2). A matching order φ of Q refers to a permutation of V_Q, determining the order in which vertices are explored during the search process. The index of vertex u in φ is denoted by $\varphi(u)$. The set of forward neighbors is defined as $N^+(u_i) = \{u_j | \varphi(u_j) > \varphi(u_i)\}$ and the set of backward neighbors is $N^-(u_i) = \{u_j | \varphi(u_j) < \varphi(u_i)\}$.

Enumerating Process. The algorithms finally conduct enumeration (line 3). The basic enumerating process first computes local candidates $C_M(u)$ (line 8), followed by extending partial matchings (line 13), and recursively carries out the computations (line 14). recursively to enumerate the results. The naive termination condition (line 5) is $i = |V_Q|$ but with techniques enumerating multiple embeddings at a time [7, 25, 95], the search depth could be reduced. The index for pruning during enumeration [4, 25, 38] is available (lines 10-11, 16) as well. Using additional indices, these methods prune some of the unnecessary backtrackings.

2.2 Related Work

2.2.1 Subgraph Matching & Enumeration. Due to the importance of subgraph matching, various algorithms [4, 6, 7, 11, 25, 34, 35, 38, 75, 81] have been proposed. The term "subgraph enumeration" has also been used in previous works [39, 42, 45, 72, 86] to refer to the same problem definition of subgraph matching. The query graphs used in their experiments are usually different from those in

subgraph matching methods in terms of vertex size and label size. In this paper, we uniformly use the term "subgraph matching" to describe the problem.

The solutions of subgraph matching can be divided into three categories, namely join-based, exploration-based, and hybrid methods. To enable parallel computing, join-based methods have been widely adopted in parallel [39, 72] or distributed manner [1, 36, 44, 45, 61, 87, 90, 96, 99]. For join-based methods, Lai et al. [46] classify join strategies into binary join [44, 45, 61, 78] and worst-case optimal join (WCOJ) [1, 3, 5, 36, 57–59, 82, 96]. Most in-memory single-threaded approaches [4, 6, 7, 11, 25, 28, 34, 38] adopt the exploration-based methods. We will introduce and compare the representative algorithms in detail in Section 3. Wang et al. [86, 87] develop a exploration-based approach in distributed system while researchers [54, 99] explore hybrid methodologies.

Comparison with existing surveys. Different from the existing surveys [47, 74] of subgraph matching, we review new enumeration techniques [4, 25, 31, 38, 95] and evaluate all the feasible combinations of different techniques while they only evaluate the original algorithms [47] or replace one technique of the original algorithms [74]. Moreover, we use more and larger real-world datasets as well as more types of synthetic datasets and label assignments. We also extend the algorithms and evaluate them on graphs with both vertex and edge labels. In addition, we adopt a novel metric EPS that can evaluate performance more unbiasedly.

2.2.2 Subgraph Search. Subgraph Search, also known as subgraph containment, is to search in a database of graphs instead of a single large graph. Given a set of graphs $D = \{G_1, G_2, \cdots, G_n\}$, subgraph search aims to find all the graphs that contain the query graph Q.

The *filtering-indexing-verification* framework is widely used in early subgraph searching efforts. The focus of such research is to design effective indices to filter out unsatisfiable instances without conducting computationally expensive subgraph isomorphism tests. One group of works [94, 101] extracts frequent features appearing in the data graph, while another group [8, 17, 23, 43] considers all features up to a bounded size. The space complexity would be $|D||\mathcal{F}|$, where $|\mathcal{F}|$ is the size of feature set, and $|\mathcal{F}|$ could be quite large.

Recently, techniques leveraging subgraph matching have emerged to tackle subgraph search efficiently without requiring indexing [52, 73]. In particular, VEQ [38] applies the same algorithmic approach to subgraph search and outperforms prior methods that rely on indexing.

2.2.3 Graph Analytic System. To facilitate large-scale graph computations, numerous graph analytic systems have been proposed, such as vertex-centric frameworks [50, 53, 69, 91], and subgraph-centric frameworks [18, 63, 79, 93]. A series of works focusing on graph mining systems have been proposed [15, 16, 20, 51, 70, 85, 100], e.g., Fractal [18], G-Thinker [93], Peregrine [30], and Sandslash [14].

These systems often provide specific subgraph matching algorithms based on the system designs.

G-thinker adopts a subgraph matching approach similar to VF2, which involves local filtering followed by enumerating all the matches. Fractal could accept a user-defined filtering function to reduce the data graph and extend vertices iteratively. Peregrine matches the core subgraph first and enumerates all the embeddings with an aggregator. Sandslash generates a matching order starting with denser sub-patterns and provides APIs `toExtend` and `toAdd` for backtracking search. SketchTree [99] is implemented in Pregel+ [92].

Table 1. Filtering techniques used in different methods.

Method	Filter Type	Number of Rounds	Filter Rule	Neighbours Usage
LDF	one-round	1	label & degree	$N_G(v)$
NLF	one-round	1	neighbor safety	$N_G(v)$
GQL	propagation	1	injective matching	$N_G(v)$
TSO	propagation	1	candidate existence	$N_T^+(u,v)$
CFL	propagation	2	candidate existence	$N_T(u,v)$
CECI	propagation	2	candidate existence	$N_{CS}(u,v)$
DPiso	propagation	3	candidate existence	$N_{CS}^{+(-)}(u,v)$
RM	join	2	the full reducer	-
VEQ	propagation	3	neighbor safety	$N_{CS}^{+(-)}(u,v)$
CaLiG	propagation	3	injective matching	$N_{CS}^{+(-)}(u,v)$

3 COMPARISON OF METHODS

3.1 Filtering Techniques

3.1.1 Overview. The filtering method is designed to identify the candidate vertex set $C(u)$ for each query vertex, with the aim of removing extraneous vertices in advance. Notably, while the worst-case time complexity remains $O(\Pi_{ui}|C(u_i)|) = O(|V_G|^{|V_Q|})$, the number of vertices can be considerably reduced in practice via candidate filtering. Ten representative filtering methods reviewed in the paper are presented in Table 1.

To categorize filtering methods, three aspects could be employed:

(1) *Filter Type*. Early filtering methods, like LDF [81] or NLF [7], employ a *one-round* filtering approach, utilizing only the local features of individual nodes. To reduce the occurrence of false positives, modern algorithms often employ *propagation* filtering techniques to achieve a more accurate candidate set. After one vertex is filtered out, the result could be utilized to refine the result of its neighbors. Specially, RM [75] employs *join* to perform filtering. In practice, the propagation method also specifies the number of rounds to traverse and refine candidates, as well as the traverse order.

(2) *Filter Rule*. The filtering rules are usually designed based on a necessary condition for subgraph matching. For local filtering, LDF [81] collects data vertex v with the same label as u such that v has a degree greater than or equal to that of u. For every label l, NLF [7] checks whether a candidate vertex v has not fewer neighbors with label l than vertex u. For propagation filtering, the filtering rules are similar in nature but are applied to the candidate set instead of the original graph, disregarding pruned vertices. For example, the neighbor-safety filter proposed by VEQ [38] could be viewed as an extension of NLF for propagation scenarios.

(3) *Neighborhood Usage.* The neighborhood usage (i.e., neighbors taken into account) for filtering is diverse as shown in Table 1. Different algorithms have designed various auxiliary data structures $\mathcal{A}$. For example, CFL and TSO only maintain tree edges (using N_T), while CECI and DPiso maintain all edges (using N_{CS}). Furthermore, for the sake of efficiency, some algorithms only consider the forward (resp. children) neighbors $N^+(u)$ or backward (resp. parent) neighbors $N^-(u)$ in a single round of propagation.

3.1.2 Latest Work. The latest work mainly focuses on designing strict filter rules to achieve a candidate set with fewer false positives.

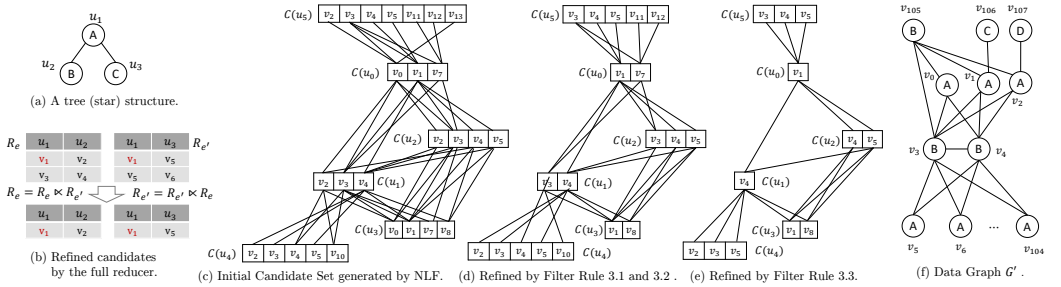


Fig. 3. Illustration of filtering methods (a)-(e) and another data graph(f).

Filter Rule 3.1. *(Candidate Existence) For each $v \in C(u)$, if there exists $u' \in N(u)$ such that $C(u') \cup N(v) = \emptyset$, remove v from $C(u)$.*

RM [75]. RM adopts a relation filter to build relations for each query edge based on labels. It decomposes Q into a set of tree-structured sub-queries, as in Figure 3(a), and prunes unnecessary tuples with the full reducer. The full reducer (Figure 3(b)) iteratively selects e whose end vertex is a leaf and e' sharing end vertex with e. The relations of e are filtered by $R_{e'} \leftarrow R_{e'} \ltimes R_e$. The pruning power of RM is competitive with the Filter Rule 3.1 [75]. The filtering techniques following Filter Rule 3.1 has the complexity $O(|E_Q||E_G|)$. For such a rule, one vertex v could be counted in multiple candidate sets, violating the injective constraints and causing false positives.

VEQ [38]. VEQ introduces a neighbor-safety filter, extending NLF on the candidate set. If $v \in C(u)$ has not fewer label-l neighbors in CS than u, $v \in C(u)$ is neighbor-safe regarding u.

Filter Rule 3.2. *(Neighbor Safety) For each $v \in C(u)$, if there exists $l \in \Sigma$ such that $|N_Q(u,l)| > |N_{CS}(u,v,l)|$, remove v from $C(u)$, where $N_{CS}(u,v,l)$ denotes the label-l neighbors of (u,v) in CS.*

VEQ implements NLF as a bit array with $4|\Sigma||V_G|$ bits to represent $|N_{CS}(u,v,l)|$ up to 4. Adopting Filter Rule 3.1 and 3.2, the complexity of neighbor-safety filter is $O(|E_Q||E_G|)$.

CaLiG [95]. The intuition of Filter Rule 3.3 is that vertex v matches u only if v's neighbors match u's neighbors. CaLiG constructs a bipartite graph for each $v \in C(u)$ and checks the existence of injective matching, similar to GraphQL [28]. GraphQL traverses $C(u)$ along an order of vertices while CaLiG propagates the state locally as it is designed for streaming graphs.

Filter Rule 3.3. *(Injective Matching) For each $v \in C(u)$, a bigraph is construct with two sets of vertices $N_Q(u)$ and $N_G(v)$, where there is an edge between $v_j \in N_G(v)$ and $u_i \in N_Q(u)$ if $v_j \in C(u_i)$. If no injective matching is found in the bigraph, we remove v from $C(u)$.*

Although Filter Rule 3.3 has the strongest pruning power, the complexity is $O(d_Q^{2.5}|E_Q||E_G|)$ as bigraph matching is required, where d_Q is the maximum vertex degree of Q.

Example 3.1.1. Consider the running example in Figure 1, the candidate set initialized by NLF is shown in Figure 3(c). No further candidates are able to be pruned by for Filter Rule 3.1, as for any $u, v, v \in C(u)$ and any $u' \in N(u)$, there exists one corresponding neighbor $v' \in N(v)$ and $v' \in C(u')$. However, for $v_2 \in C(u_1)$, v_2 has only one neighbor labeled A, but two neighbors labeled A are needed for u_1. Therefore, v_2 is not neighbor-safe and could be removed from $C(u_1)$. The candidate set refined by Filter Rule 3.1 and 3.2 is shown in Figure 3(d). For $v_7 \in C(u_0)$, although v_7 is neighbor-safe, its neighbor v_3 is used twice in $C(u_1)$ and $C(u_2)$, violating the injective constraints. By Filter Rule 3.3, v_7 could be pruned from $C(u_0)$ and the final candidate set (Figure 3(e)) is much more tighter, with fewer false positives.

3.2 Matching Orders

3.2.1 Overview. The ordering method is to generate a matching order to explore the search space. The ordering algorithms could be roughly categorized into three groups.

(1) *Data-Independent Order.* Data-independent order methods solely utilize the information of the query graph to generate orders. RI [9] first selects the vertex with the largest degree as the start vertex, and iteratively selects the vertex with the most backward neighbors. RM [75] constructs a density tree for the query graph, and arranges the matching order starting from the densest part to the sparsest part. Data-independent order is usually efficient and empirically effective. RI is the recommended ordering technique for sparse queries as discussed in [74].

(2) *Data-Dependent Order.* Data-dependent order methods are commonly adopted by most algorithms, which leverages both query and data graphs to generate an effective matching order.

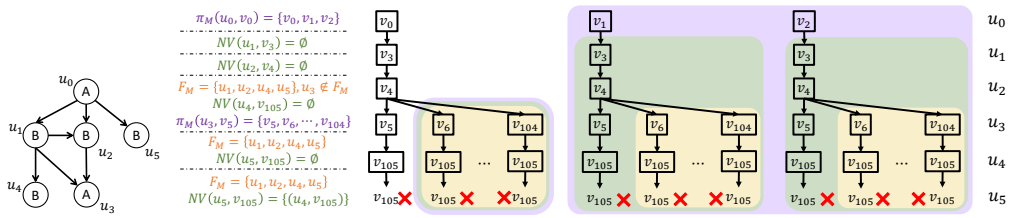
The first consideration is the selectivity of labels. For simplicity, we denote the occurrence of a label in data graph as $|L_G(u)| = |\{v \in V_G | L_Q(u) = L_G(v)\}|$. VF2++ [35] first picks the vertex with least $|L_G(u)|$ and iteratively selects the vertex with the most backward neighbors. QuickSI [65] assigns weight to the edges by selectivity and selects the unvisited edge (whose one vertex is selected) with the minimum weight.

Since the filtering method is adopted, the selectivity for query vertex u could be more accurately estimated by the size of the candidate set $|C(u)|$. GraphQL [28] iteratively selects the query vertex with the fewest candidates (i.e., the minimum $|C(u)|$) and CECI [6] selects the vertex with the minimum $\frac{|C(u)|}{d(u)}$ as a start. CFL [7] proposes the core-forest-leaf decomposition and a path-based ordering method is proposed for the matching order of core vertices. CFL generates a BFS tree, enumerates all root-to-leaf paths, and selects the path with the minimum estimated number of embeddings.

(3) *Dynamic Order.* In a large data graph, the distribution of vertex labels could be unbalanced in various regions within one single graph. Therefore, a fixed matching order could be sub-optimal. TSO [26] partitions the search space into candidate subregions, defined as the search subtree rooted at assignment (u, v). For each subregion, TSO generates a matching order based on the selectivity in the subregion. DPiso [25] proposes an adaptive order method to adjust the order in each backtracking step. Two kinds of orders are available. Candidate-size order selects an extendable vertex u with minimum $|C_M(u)|$, while path-size order selects an extendable vertex u with minimum estimated number of path embeddings.

3.2.2 Latest Work. **VEQ [38].** CFL [7] proposes leaf decomposition (as discussed in Section 3.3.3), which has been widely adopted by subsequent algorithms like DPiso. VEQ follows the candidate-size order of DPiso but generates a new adaptive order. In CFL and DPiso, the leaf decomposition needs to postpone the matching of leaf vertices (i.e. degree-one vertex) to the end of the matching order. However, when the failures happen in the matching of leaf vertex, more invalid backtrackings would be wasted. Therefore, the dynamic order of VEQ checks whether the size of remaining candidates for a degree-one extendable vertex u is less than or equal to the needed size, to detect the failure in advance.

Example 3.2.1. Let us consider query graph Q in Figure 1(a) and data graph G' in Figure 3(f). There is no embedding of Q in G' as there are insufficient vertices labeled as B to assign to the leaf vertices u_4 and u_5. However, if we postpone the matching of the leaf vertices with the matching order $(u_0, u_1, u_2, u_3, u_4, u_5)$, we need to traverse the 100 candidates from v_5 to v_{104} to identify the failure. For the adaptive order of VEQ, after assigning v_3 to u_1 and assigning v_4 to u_2, there is only one candidate for leaf vertex u_4, so u_4 is selected instead of u_3. Since no candidate exists for u_5, and the failure has been detected. and the invalid traversal can be avoided.



(a) q_D for DPiso (b) Search tree for Q and G'.

Fig. 4. Illustration of pruning during enumeration.

RM [75]. RM considers the cardinality estimation of sub-queries challenging and thus turns back to the graph structure of Q. CFL proposes core-forest-leaf decomposition and starts the enumeration from the dense part (i.e., core vertices). RM extends the idea and takes advantage of the dense sub-structures in core vertices. It constructs a density tree with nucleus decomposition [67], which could find dense subgraphs with a multi-level hierarchy. RM then starts searching in the densest part and ends at the sparsest part.

3.3 Enumerating Paradigms

As described in Algorithm 1, the existing techniques for improving the embedding enumeration can be classified into three categories: (1) Local candidate computation. (2) Pruning during enumeration. (3) Enumerating multiple embeddings at a time.

The local candidate computation (line 8) refers to the approach of computing extendable data vertices at each backtracking step. A variety of algorithms and data structures are proposed [74]. For subgraph matching, the number of backtrackings could be $O(|V_G|^{|V_Q|})$, and backtracking would suffer from numerous duplicates or futile backtracking. Recently, more efforts try to reduce the number of backtrackings instead of accelerating each individual backtracking itself. Generally, there are two ways to reduce the number of backtrackings. One is to perform pruning during enumeration (lines 10-12, 16) to avoid duplicate or futile backtracking, while the other one is enumerating multiple embeddings at a time, stopping the searching process earlier (lines 5-6).

3.3.1 Local Candidate Computation. For local candidate computation, without auxiliary data structure, we have to loop all the vertices in $N(M(u.p))$ (QSI, RI) or $C(u)$ (GQL), and then check the existence of edges between v and $M(u')$ for all $u' \in N(u)$. The supplementary data stored within $\mathcal{A}$ would prove to be beneficial. It allows the algorithm to directly retrieve the neighbors of v in $C(u)$. CFL maintains tree edges while CECI and DPiso maintain all edges and compute the local candidates with the set intersection.

3.3.2 Pruning during Enumeration. Pruning during enumeration is one approach to avoid duplicate backtracking by eliminating duplicate search subtrees. It encompasses two aspects: (1) recording past failures to prevent their recurrences, and (2) reusing prior successful results to avoid redundant backtrackings. To record the failures, the concept of nogood is introduced [4, 71].

Definition 3.1 (Nogood). *Set of assignments $D \subset V_Q \times V_G$, D is a nogood, iff no full embedding $M : V_Q \rightarrow V_G$ that $D \subset M$.*

Nogood refers to a partial embedding that does not appear in any results. By identifying nogoods, the invalid search path can be safely terminated.

Failing Set. Failing set is proposed in DPiso [25]. DPiso has built a rooted DAG q_D by BFS, capturing the dependence on assignments. Failures are classified into two classes: (i) *conflict-class*, where one data vertex is used twice in a single embedding; (ii) *emptyset-class*, where the local candidates for u to extend is empty. For (i), failing set is $F_M = \text{anc}(u) \cup \text{anc}(u')$ while for (ii), $F_M = \text{anc}(u)$, where $\text{anc}(u')$ denotes the set of all ancestors of u in q_D including u itself. The failing set F_M of an inner node in the search tree is updated from its children F_{M_i}. Given failing F_M, for a search node (i.e., partial embedding) in the search tree (u, v), if $u \notin F_M$, then all siblings of node M are redundant. In other words, the nogood discovery adopted in failing set is $D = \{(u, M(u)|u \in F_M\}$, and DPiso employs backjumping until the assignment in the nogood changes.

Example 3.3.1. Consider the query graph Q in Figure 1(a) and the data graph G' in Figure 3(f), the DAG generated for Q is shown in Figure 4(a). The partial mapping in (u_5, v_{105}) fails and belongs to conflict-class and $F_M = \text{anc}(u_4) \cup \text{anc}(u_5) = \{u_0, u_1, u_4, u_5\}$. The failing set is updated in a bottom-up fashion. Because $u_3 \notin F_M$, the siblings of (u_3, v_5) are redundant. However, since failing set is discarded when backjumping has been done, the same process is still needed for partial embedding involved (u_0, v_1) and (u_0, v_2). The space pruned is illustrated in yellow in Figure 4(b).

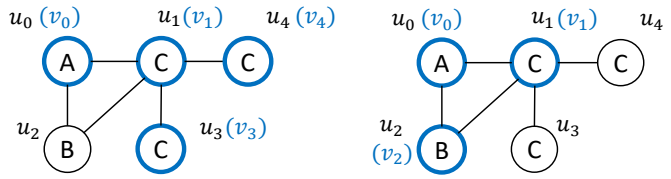
Guard-based Pruning. The failing set of DPiso would be discarded after it has been used for backjumping. GuP [4] proposes guard-based pruning to reuse a discovered nogood for pruning multiple times, at the cost of extra memory consumption. The *nogood guard* $NV(u_i, v)$ in GuP is constructed based on deadend mask. The deadend mask, denoted by K, captures the adjacency constraint related to the failure, based on the concept of *local candidate-vertex set*. The *local candidate-vertex set* of u_i under M, is a set of $v \in C(u_i)$ that holds $v \in N_G(M(u_j))$ for all $u_j \in N_Q(u_i)$. Also, GuP proposes the *reservation guard* to prevent conflict-class failure.

Example 3.3.2. The partial matching on (u_5, v_{105}) has an injectivity conflict, and the deadend mask is $K_{(u_5,v_{105})} = \{u_4, u_5\}$. With the bottom-up updating, we have $K_{(u_4,v_{105})} = \{u_4\}$, $K_{(u_3,v_5)} = \emptyset$ and finally $K_{(u_1,v_3)} = \emptyset$. Since $NV(u_1, v_3) = \emptyset$ is the subset of any M, the subtree rooted at (u_1, v_3) could be pruned. The nogood guard on (u_1, v_3) is reusable for M beginning with (u_0, v_1) and (u_0, v_2) as well. In Figure 4(b), the space pruned is marked in green.

Dynamic Equivalence Class. While failing set and guard-based pruning only record the failure, VEQ [38] proposes dynamic equivalence to remove the equivalent subtree, the failed ones as well as the successful ones. The idea of dynamic equivalence class may be inspired by the concept of equivalence class. Due to the exclusive focus on the nodes within the candidate set, the dynamic equivalence class is defined over the candidate set. For $v_i, v_j \in C_M(u)$, v_i and v_j share neighbors if v_i and v_j have common neighbors in $C_M(u_j)$ for every $u_j \in N_G(u)$. After excluding some exceptional cases, v_i and v_j are symmetric to each other, denoted by $\pi_M(u, v_i) = \pi_M(u, v_j) = \{v_i, v_j\}$. Thus, under the search subtree of M, for any embeddings containing v_i, a symmetric embedding containing v_j can be generated directly. For failure, dynamic equivalence class could be considered as symmetricity-based nogood discovery. The equivalence can be only used for siblings of M, and not reusable for other search subtrees.

Example 3.3.3. It is obvious that vertices v_5 to v_{104} share common neighbors, and $\pi_M(u_3, v_5) = \{v_5, v_6, \cdots, v_{104}\}$. The subtree rooted at $M \cup (u_3, v_5)$ is visited and no embedding is found. This implies that the subtree rooted at all data vertices in $\pi_M(u_3, v_5)$ is identical, and there is no need for further exploration. In addition, although v_0, v_1, v_2 have different neighbors in G', they share common neighbors in the candidate set after filtering, $\pi_M(u_0, v_0) = \{v_0, v_1, v_2\}$. Therefore, the subtree rooted at (u_0, v_1) and (u_0, v_2) can be pruned. In Figure 4(b), the space pruned is marked in purple.

3.3.3 Enumerating Multiple embeddings at A Time. Instead of backtracking until all query vertices are assigned, there are ways to terminate the search in advance and produce multiple embeddings.
One-step ahead. One-step ahead is a fundamental improvement in the basic backtracking framework. During the final step of backtracking, for each unvisited data vertex in the local candidates, there exists a complete embedding. The termination condition in Algorithm 1 is $i = |V_Q| - 1$. For example, as shown in Figure 5(a), all the query vertices except u_2 have been matched. We directly intersect u_3's and u_6's neighbors in candidate sets, and one embedding is generated for each local candidate.



(a) One-step ahead (b) Leaf enumeration

Fig. 5. Illustration of one-step ahead and leaf enumeration.

Leaf enumeration. For a leaf vertex, i.e., a degree-one vertex, the matching depends on its only neighbor. TSO [26] permutes the vertices in the same neighbor equivalent class (NEC) to generate embeddings at a time. CFL [7] proposes leaf decomposition, and the related leaf-matching algorithm to effectively enumerate the Cartesian product. Compared to TSO, CFL not only considers NEC vertices but also puts all leaf query vertices with the same label in the label class. The two leaf vertices u_4 and u_5 in Figure 1(a) are not in the same NEC but in the same label class. DPiso [25] also takes the leaf decomposition. VEQ [38] finds NEC among all degree-one vertices in Q, and merges the vertices in q_D. The query vertices are decomposed into the set of degree-one vertices and the remaining set $V_{Q'}$. The worst-case complexity for subgraph matching is $|V_G|^{|V_{Q'}|}$ instead of $|V_G|^{|V_Q|}$. For example, the leaf vertices u_3 and u_4 are unmapped in Figure 5(b), embeddings could be obtained by simply permuting u_6's neighbors in candidate sets.

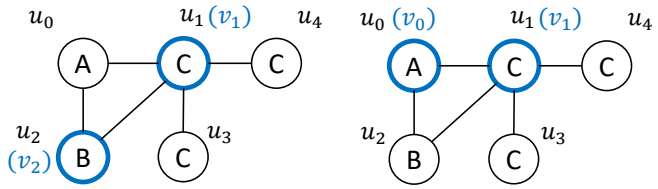


Fig. 6. Two kernel-and-shell decompositions.

Kernel-and-Shell Search. Kernel-and-Shell Search (KSS) proposed by CaLiG [95] decomposes the query vertices into a kernel vertex set V_{kernel} and a shell vertex set V_{shell}. Kernel vertex set is a connected vertex cover of Q, and the shell vertex set is the complementary set of kernel vertex, which is an independent set. Therefore, no adjacency constraints exist between shell vertices, they could be matched directly. For example, two kernel-and-shell decompositions are depicted in Figure 6, where the part in blue is the kernel set. KSS only conducts backtrackings on kernel

Table 2. Techniques to Evaluate.

Stage	Techniques to Evaluate
Filter	fLDF [81], fNLF [7], fGQL [28], fTSO [26], fCFL [7], fCECI [6], fDPiso [25], fRM [75], fVEQ [38], fCaLiG [95]
Order	oQSI [65], oGQL [28], oTSO [26], oCFL [7], oDPiso [25] oCECI [6], oRI [9], oVF2P [35], oVF3 [11], oRM [75]
Enum	eEXPL [81], eLFTJ [83], eGQL [28], eQSI [65], eVF3 [11], eDPiso [25], eCECI [6], eRM [75], eVEQ [38], eKSS [95]

Table 3. Data graphs with vertex labels.

Dataset	Abbreviation	$\|V\|$	$\|E\|$	d_G	Core Number	Type
Figeys	*fg*	2,239	6,432	5.7	10	Protein
YeastS	*ys*	2,361	7,182	6.1	10	Protein
Human	*hm*	4,674	86,282	36.9	148	Protein
HPRD	*hp*	9,303	34,998	7.5	14	Protein
citeseer	*cs*	3,279	4,552	2.8	7	Citation
WordNet	*wn*	146,005	656,999	9.0	31	Lexical
Standford	*sf*	281,903	1,992,636	14.1	71	Web
DBLP	*db*	317,080	1,049,866	6.6	113	Collab.
Twitch	*tw*	168,114	6,797,557	80.9	149	Social
Youtube	*yt*	1,134,890	2,987,624	5.3	51	Social

vertices, and enumerates all embeddings for shell vertices at a time. The worst-case complexity of KSS is reduced to $O(|V_G|^{|V_{kernel}|})$.

Since CaLiG is originally designed for streaming graphs, in order to make it applicable to all input orders, we adjust the decomposition of kernel and shell. Given a matching order, the query vertex u whose $N_Q^+(u) = \emptyset$ is added into the shell vertex set.

4 EXPERIMENTAL EVALUATION

4.1 Experimental Setting

Techniques to Evaluate. We select ten representative techniques for each stage (i.e., filter, order, and enumeration), as shown in Table 2, and *the total number of feasible algorithm combinations amounts to 534*. Note that several techniques are not able to combine with other techniques since they may highly depend on specific data structures. We carefully re-implement all these techniques mentioned, except those covered by [74]. For ease of presentation, prefixes are added to indicate specific components of a method: 'f' for filter, 'o' for order, and 'e' for enumeration. For example, fVEQ represents the filtering technique of VEQ.

Data Graphs. We select 14 typical real-world graphs across different domains. To minimize the influence of human bias on the selection of label size, we conducted experiments on 10 datasets in Table 3 with label sizes of 15, 30, 45, and 60. The vertex labels are randomly assigned following the previous work [4, 74]. As shown in Table 4, we also collect 4 datasets with real vertex and edge labels. For further details, please refer to Section 4.3.

We use ER model [13] and RMAT model [12, 37] to generate random graphs and power-law graphs, respectively. For each model, the default settings are $|V| = 1M$, $|E| = 5M$, and $|\Sigma| = 30$. In

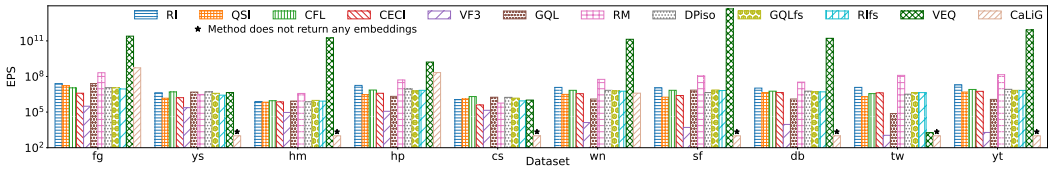


Fig. 7. The overall performance of 12 original algorithms.

order to evaluate scalability, we vary the number of vertices $|V|$ at 0.05M, 0.1M, 0.5M, and 1M, the number of edges $|E|$ at 5M, 10M, 15M, and 20M, and the label cardinality $|\Sigma|$ at 15, 30, 45, and 60.

Query Graphs. To generate query graphs, we follow the approach as described in previous studies [4, 25, 74]. The sampling method performs a Metropolis-Hastings random walk [27] on data graphs and extracts the induced subgraphs as queries. The size of query graphs ranges from 10, 20, 30, 40, and 50, and each query set contains 1000 query graphs. We generate queries for graphs with real edge labels, each consisting of 100 queries. For real queries on DBpedia, we adopt the LC-QuAD workload [80] in the experiment.

Evaluation Metrics. We measure the effectiveness and efficiency of each technique with the following metrics.

- *Average candidate size.* The average candidate size after filtering, i.e., $\sum_{u \in V_Q} |C(u)|/|V_Q|$, denoting the effectiveness of filter.
- *Embeddings per seconds (EPS).* Considering both the elapsed time and the number of reported embeddings, EPS is defined as the average number of embeddings returned per second and provides a consistent evaluation of algorithm performance across time and output size limitations. Since EPS is an efficiency metric (like speed), it is important to note that a larger EPS score does not necessarily correlate to a larger number of results.

Instead of stopping the search upon finding 10^5 embeddings as previous methods [4, 25, 38, 74], we have not imposed any maximum limit on the number of embeddings. In our opinion, subgraph matching could yield a large number of embeddings, easily exceeding 10^5 or even 10^8. Limiting the search to 10^5 embeddings may only cover a small fraction of the entire search space. Section 4.5 will provide detailed insights into how output limits can affect the performance of the algorithms. Due to the large number of queries that need to be executed, we set a timeout of one second and use EPS as the evaluation metric. It is worth noting that EPS optimizes for the average case rather than the worst case Reporting the response time by limiting the output size is likely to skew the metric significantly if a single worst-case query takes up most of the total time. Contrarily, EPS remains relatively stable even in the face of some worst-case queries.

Experiment Environment. All the algorithms are implemented in C++ and evaluated on a Linux Server equipped with Intel(R) E5-2596v4 CPU @2.2 GHz and 128G RAM. The source codes will be available at: https://github.com/JackChuengQAQ/SubgraphMatchingSurvey.

4.2 Performance of Original Algorithms

The overall performance (i.e., EPS) of the original algorithms, (i.e., RI [9], QSI [65], CFL [7], CECI [6], VF3 [11], GQL [28], RM [75], DPiso [25], GQLfs [74], RIfs [74], VEQ [38], CaLiG [95]) is shown in Figure 7. For methods that return no results, we still plot a placeholder and mark it with '★' on the top. CaLiG completes the queries on three graphs as its filter takes too much time (up to 1 second sometimes). VF3 performs the worst, possibly because it is designed for small graphs. Most methods yield comparable performances, while RM and VEQ demonstrate significant advantages on most graphs. The findings support the notion that significant improvements can

Table 4. Data graphs with real vertex and edge labels.

Dataset	Abbreviation	$\|V\|$	$\|E\|$	$\|\Sigma\|$	$\|\Sigma^E\|$
Wordnet18	*wm*	40,943	75,770	4	18
FreeBase15k	*fb*	14,951	260,184	1	1345
Telecom	*tc*	170,380	8,900,000	398	300
DBpedia	*dp*	7,739,704	21,429,823	259,623	633

be achieved by enhancing the backtracking framework, considering that backtracking is a crucial factor contributing to exponential time complexity in computations. However, we notice that VEQ exhibits an unusually limited performance on the dataset *tw*. Due to the high average degree of dataset *tw*, the phenomenon may be attributed to the relatively large average candidate size, coupled with the limited number of equivalent classes on the dataset. Consequently, the cost of finding equivalent classes would be expensive, and the benefits fail to justify the expense.

Significance Test. The performance of the original methods is assessed using the non-parametric Friedman test [21], yielding p-value $p = 6.21\times10^{-9}$, thus indicating a markedly significant difference in method efficiencies. Significance tests are also conducted on each method effectiveness' difference across all datasets. For each method, the test yields very significant differences with $p < 10^{-9}$, which supports that different methods may have varying performance across different datasets. The post-hoc Nemenyi test [56] is also conducted for paired comparison. The performance on most pairs of datasets is significantly different.

4.3 Performance on Graphs with Edge Labels

Despite most previous studies focusing on vertex-labeled graphs, it is interesting to explore the performance on graphs with both vertex and edge labels. Consequently, we extend the reviewed methods to accommodate queries with both vertex and edge labels, by checking vertex and edge labels at the same time. We select four datasets with real labels for the evaluation, as presented in Table 4. Dataset *tc* is telecom network and *dp* is knowledge graph. Datasets *wm* and *fb* are sourced from [10].

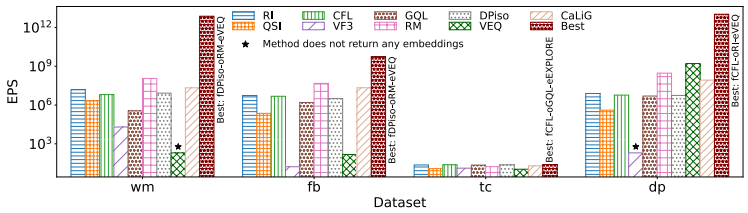


Fig. 8. Performance on graphs with edge labels.

The performance of both the original algorithms and the best-performing combination on edge-labeled graphs are showcased in Figure 8. In comparison to graphs without edge labels, the differences between the original methods have relatively narrowed. However, the best combination far surpasses all original methods, demonstrating the necessity of studying the interplay between techniques. VEQ does not perform well on datasets *wm* and *fb*, due to the additional requirement to check edge labels in the neighbor-safety inspection within the fVEQ technique. As a comparison, across datasets *wm*, *fb*, and *dp*, the enumeration technique eVEQ is used in the best-performing combinations. CaLiG exhibits a good performance across all datasets, because fewer bipartite graph

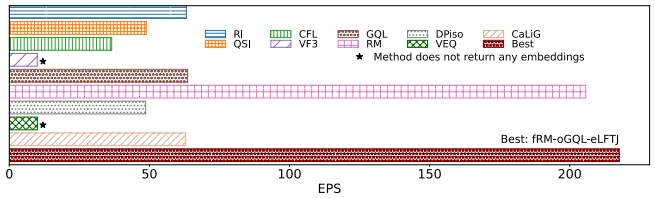


Fig. 9. Performance on real queries.

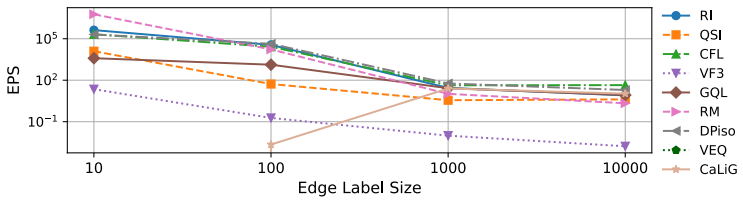


Fig. 10. Evaluation of edge label size.

checkings are required benefitting from the introduction of edge labels and higher selectivity. The inferior performance of all methods on dataset *tc* is due to the degree distribution of *tc* is nearly uniform, not conforming to the power-law property.

Evaluation of Real Queries. We extract real query workloads from LC-QuAD [80]. Given that LC-QuAD pertains to natural language questions on knowledge graphs, the queries typically consist of around 3 to 5 vertices, and the number of embeddings tends to be relatively small. Owing to the small size of the query graphs, most complex filtering and enumerating techniques yield little benefit. The enumeration stage could swiftly end due to the limited search depth. Consequently, the RM method, which directly employs a join-based approach for filtering, exhibits the best performance.

Evaluation of Edge Label Size. We investigate the effect of edge label size on algorithm performance by varying edge labels from 10 to 1000. The results on *wm* are presented in Figure 10. With the increase in edge label size, most algorithms experience a decrease in EPS due to the reduced number of embeddings and relatively constant filtering time. This results in an increased time overhead per embedding. However, there is an exception in the case of CaliG algorithm. The substantial cost of fCaLiG degrades its performance when dealing with graphs of small edge label size. Nevertheless, as the edge label size increases, the performance of CaliG improves.

4.4 Enable A Single Solution for Automorphism

The methods following Definition 2.2 may generate multiple embeddings for a single automorphism group. There might be an exponential number of embeddings that are symmetrically equivalent, potentially leading to computational inefficiency. To explore the impact of automorphisms, we extend the original algorithms based on the method articulated in ISMAGS [29]. The extension entails an analysis of symmetries to derive constraints which are then scrutinized during the search phase to effectively break symmetry.

We select datasets *hp*, *cs*, *db*, and *tw* with $|\Sigma| = 15$, and collect 100 query graphs which has non-trival antomorphism. The experimental results are presented in Figure 11. It is observed that the algorithms RI, CFL, and DPiso exhibit comparable and commendable performance. VF3's performance on dataset *tw* is less optimal due to the large average degree. Upon observing DPiso's

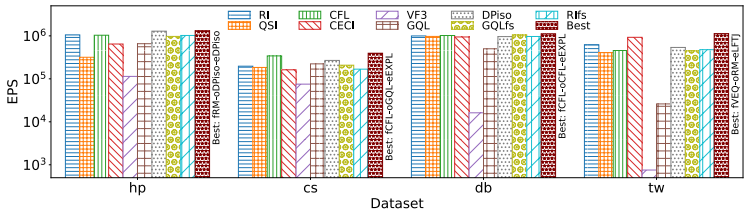


Fig. 11. Enabling a single solution for automorphism.

performance, it appears that the effectiveness of the failing set method has decreased since the introduction of symmetry-breaking constraints.

4.5 Impact of Output Limit

Previous studies often evaluate subgraph matching algorithms with a fixed output limit (e.g., 10^5). However, this output limit can significantly influence the relative performance and rankings of methods. We vary the output limits from 10^5 to 10^9 by setting $|\Sigma| = 15$ and $|V_Q| = 10$. The results on dataset *dp* are shown in Figure 12, where each line represents the efficiency rankings of one technique combination under various output limits. Figure 13(a1) depicts the effect of filter techniques, in which the combinations that adopt the same filter technique are plotted in an identical color. Similarly, Figures 13(b1) and 13(c1) depict the effect of ordering and enumeration techniques, respectively. Figures 12(a2), 12(b2), and 12(c2) illustrate the ranking changes of each filter, ordering, and enumeration technique under its respective best-performing combination.

We can observe a significant impact of the output limit on algorithm rankings. As illustrated by the bold line in Figure 12(a1), the top-ranked algorithm at output limit 10^5 drops to near the bottom at 10^9, while the top-ranked algorithm at output limit 10^9 is ranked around 300th place at output limit 10^5.

Among the three stages, the impact of enumeration still appears to be the greatest, with very clear clustering observed in Figure 12(c1). As shown in Figure 12(c2), at lower output limits, eEXPL performs the best. As the output limit increases, its ranking declines, while eVEQ and eKSS rise, ultimately becoming the top performers. For tasks requiring a limited number of embeddings, eEXPL swiftly identifies embeddings without additional data structures or computations. However, as the required embeddings increase, algorithms, such as eVEQ and eKSS, could avoid duplicate computations and efficiently retrieve a large number of embeddings.

In Figure 12(a2), we observe significant variation in performance among different filtering techniques initially. However, as the output limit increases, the influence of filtering cost diminishes, resulting in a convergence of performance. In Figure 12(b2), many ordering techniques contribute to higher rankings of different combinations.

In order to further illustrate the relative consistency of the proposed metric (EPS), we choose the top 50 algorithms with the largest variance in the output limit experiment and evaluate them on the same queries with different time limits (1s, 5s, 10s, 30s, 60s). As depicted in Figure 13, the EPS rankings exhibit small fluctuations, indicating a consistent pattern or trend. This demonstrates the effectiveness of using EPS as a metric for evaluating the algorithms.

4.6 Interaction Effect of Combinations

Previous works left the interaction of algorithm combinations to be further explored. Our experiments, however, underscore the considerable importance of these interaction effects.

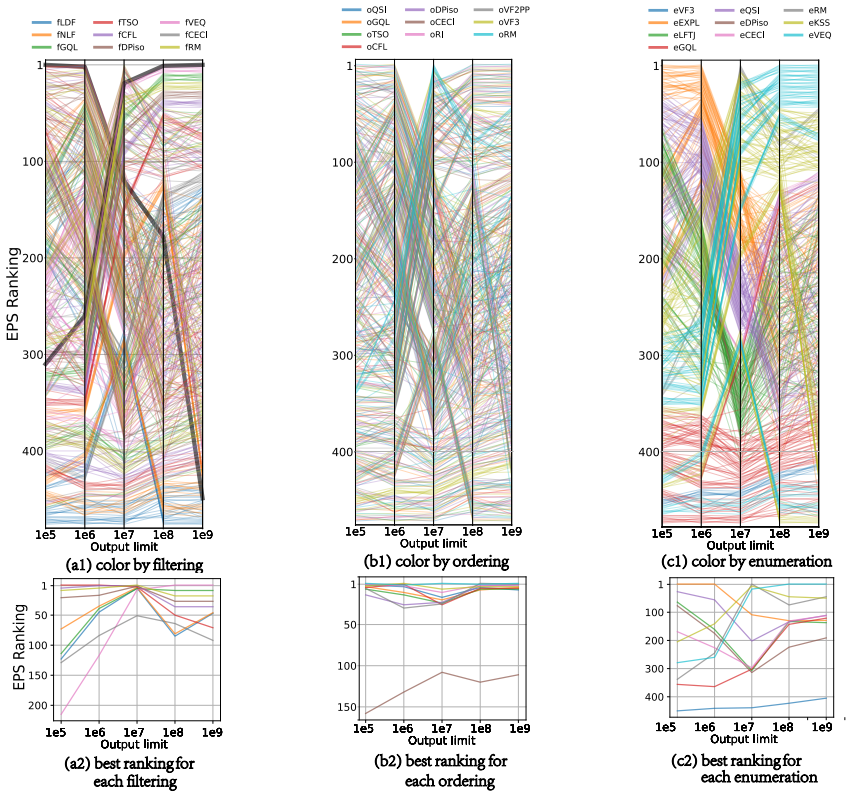


Fig. 12. Efficiency rankings under different output limits.

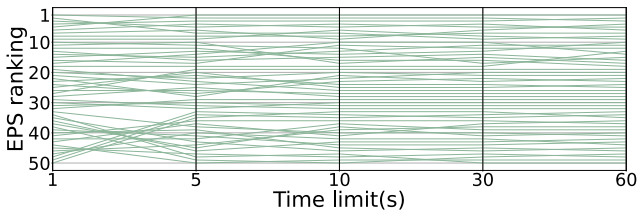
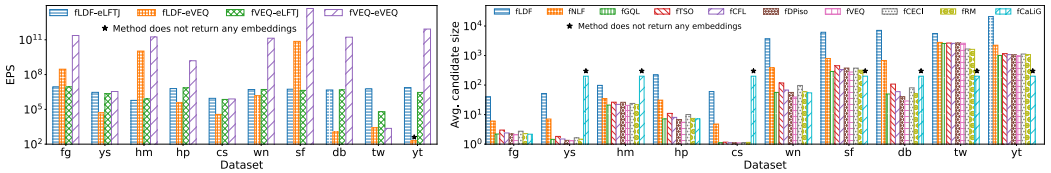


Fig. 13. EPS rankings under different time limits.

By fixing the ordering method as RI, we combine filters fLDF and fVEQ with enumerating methods eLFTJ and eVEQ, exhaustively. As shown in Figure 14(a), the impact of filter replacement for eLFTJ is not significant, whereas the impact is substantial for eVEQ. When eVEQ is combined with fVEQ, the performance is significantly improved. However, in datasets like *cs*, *db*, and *tw*, the performance of eVEQ combined with fLDF is even worse than that of fLDF-eLFTJ, In fact, interaction effects exist widely between algorithm combinations. As demonstrated in Figures 15, 16, and 17 for evaluations in Sections 4.7, 4.8, and 4.9 respectively, the optimal combinations of particular techniques vary across datasets, indicating a better synergy between the techniques and their optimal combinations. The factors underlying the emergence of interactions will be further discussed in the following subsections.



(a) An example of the interaction effect.

(b) The average candidate size.

Fig. 14. Illustrations of interactions and candidate size.

4.7 Effect of Filtering Techniques

To investigate the pruning power, we present the average candidate size of each filtering technique as shown in Figure 14(b). The inferior pruning power of one-round filters (fLDF and fNLF) becomes evident when compared to propagating filters, as they often generate one or two orders of magnitude more candidates. Overall, the performance of propagation methods is relatively similar. Due to diminishing marginal utility, it is unlikely that more complex filters can significantly prune a much larger number of candidates. Due to the incorporation of neighbor-safety filter, fVEQ exhibits a slight improvement in pruning power compared with fDPiso. The experimental results indicate that fGQL and fCaLiG exhibit favorable performance on graphs *fg* and *hp* because of the powerful Filter Rule 3.3. However, they still fail to beat fVEQ in pruning power, as they lack a well-designed filter order. Moreover, the similar performance of fGQL and fCaLiG suggests that the benefits of repeatedly refining by injective matching may be limited.

The performance of methods varies across datasets. On *cs* and *ys*, propagation filters significantly outperform one-round methods (fLDF and fNLF) while on datasets *tw* and *yt*, multiple rounds of refinement fail to prune more candidates than fNLF.

To facilitate a fair comparison of different filtering techniques, we evaluate each filter under its best combination across all datasets. As shown in Figure 15, the corresponding best combinations are indicated above each bar in the figure. The inferior performance of fLDF and fNLF could be attributed to their limited pruning power. Although we observe some gap in pruning power among the propagation filters in Figure 14(b), the performance differences in their EPS are negligible after applying the optimal combinations. Some false positive candidates, that are not pruned by filters, can be effectively captured by the technique of pruning in enumeration.

Different algorithms have their own strengths and weaknesses, and therefore, the optimal combination of filter techniques varies across different graphs. Considering the enumerating methods, the optimal selection is eKSS for *cs* and *ys* graphs, eRM for *tw* graphs, and eVEQ for the remaining datasets. Furthermore, the selection of ordering methods can exhibit greater variability.

4.8 Effect of Ordering Techniques

We evaluate the performance of ordering techniques under the optimal combination of filtering and enumerating techniques. The results are presented in Figure 16. We do not include eVEQ in the optimal selection of ordering techniques, since it employs a dynamic order. Overall, data-dependent techniques oCECI, oCFL, and oDPiso underperform other techniques, due to the challenges in accurately estimating cardinality. The data-independent technique oRM demonstrates strong performance on most datasets. However, it faces challenges when dealing with dataset *cs*. As shown in Table 3, *cs* has the smallest average degree (2.8) and core number (7), indicating its high sparsity. Correspondingly, the candidate size for *cs* (as depicted in Figure 14(b)) is also very small. In such scenarios, the influence of the topology structure of query graphs on the matching order becomes

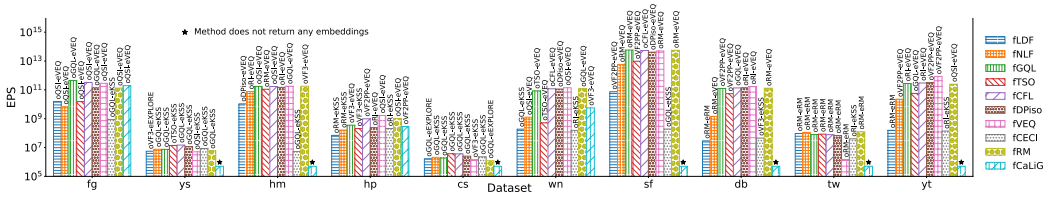


Fig. 15. Performance of filtering with the best combination (on average over all label and query sizes).

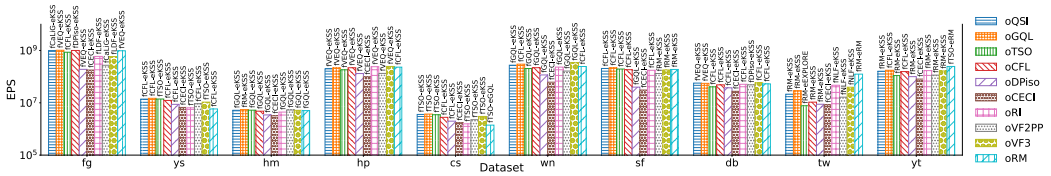


Fig. 16. Performance of ordering with the best combination (on average over all label and query sizes).

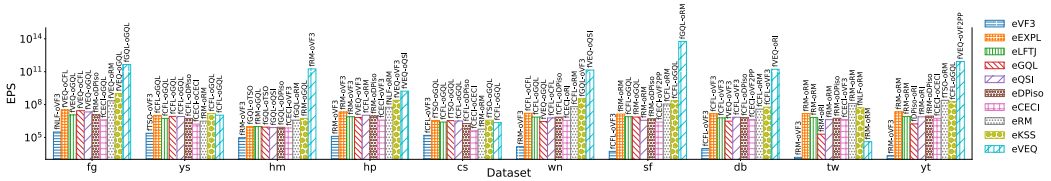


Fig. 17. Performance of enumerating with the best combination (on average over all label and query sizes).

less significant compared to the selectivity affected by the data graph. This leads to data-dependent methods outperforming data-independent ones. On the other hand, for datasets like *tw* that have larger values for average degree and core number, oRM consistently delivers superior performance.

In terms of the optimal choices, all ordering methods use eKSS as the optimal enumeration method. The only exception is oRM, which may occasionally select the original combination (oRM-oRM-oRM). It is observed that eKSS displays a strong reliance on the ordering method utilized, as the input order directly influences the kernel-and-shell decomposition. It demonstrates that data-independent methods tend to focus more on the graph structure, often leading to larger shell sets. In conclusion, both the graph structure and candidate size are crucial for developing a good order for subgraph matching.

4.9 Effect of Enumerating Techniques

We evaluate the performance of each different enumerating technique under its best combination of filtering and ordering techniques. Figure 16 presents experimental results.

It is observed that eEXPL, eLFTJ, eGQL, and eQSI are competitive with each other. Recently proposed methods like eRM, eVEQ, and eKSS have demonstrated varying degrees of advantages over the other algorithms. Compared to eLFTJ, which is used as the baseline, RM achieves an improvement of approximately 10x speedup. eKSS employs a native backtracking search for kernel vertices, while enumerating all shell vertices at a time. It exhibits better performance and approximately 24x speedup compared to eLFTJ. Under certain queries, kernel-and-shell decomposition can be more extensive compared to the core-based decomposition used in eRM. For example, for a

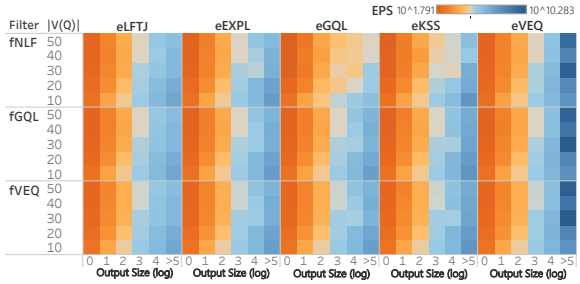


Fig. 18. Varying the output size and the query size.

2-core query, the core vertices encompass the entire query, and yet kernel-and-shell decomposition remains applicable.

Technique eVEQ introduces dynamic equivalence class detection during backtracking search to avoid searching through duplicate subtrees. It yields remarkable performance boosting, with an improvement of 3∼8 orders of magnitude compared to eLFTJ, except on graphs *cs*, *tw*, and *ys*. However, the trade-off for using eVEQ is that its performance may be inferior to that of eLFTJ on certain graphs. These enumerating methods have shown promising results and suggest modifying the backtracking framework may lead to further improvements in subgraph matching.

When considering the optimal combinations for enumerating methods, it is worth noting that the optimal selection of filtering and ordering methods still vary across different graphs. Even when we initially claim that the eVEQ enumeration heavily depends on the filtering method used, it does not always choose the tighter filter like the eVEQ filter. At times, it is not necessary to employ complex filtering rules to sufficiently reduce the candidate set.

4.10 Evaluation of Output and Query Size

Figure 18 presents the performance of the algorithms under different output sizes and query sizes, on the dataset *db* with a label size of $|\Sigma| = 60$. DP stands as an abbreviation for DPiso. We use a fixed ordering technique oRI and explore combinations of filtering and enumerating techniques.

As shown in Figure 18, different methods exhibit similar patterns. Given the high selectivity by $|\Sigma| = 60$, even the simple fNLF has not demonstrated any significant disadvantages. It is observed that EPS generally increases as the output size grows, which is expected since a larger output size often implies that the results are easier to find. In addition, a larger output size also suggests that enumeration techniques that enhance the enumeration framework would exhibit a more significant advantage. Such techniques, like eKSS and eVEQ, may achieve more opportunities to enumerate multiple results at once or reuse the results maintained from the equivalent class. In terms of the query size, EPS tends to decrease as the query size grows. The only exception is the performance of eVEQ when the output size exceeds 10^5.

4.11 Evaluation of Graph Meta-feature

Treewidth is considered a significant factor affecting the efficiency of graph queries [2]. Figure 19 presents the results of typical technique combinations under varying tree widths, on graph *db* with $|\Sigma| = 15$ and $|V_Q| = 50$. The top label denotes the class of enumeration techniques, the bottom label signifies the filtering methods, and the left one represents the orderings. The treewidth fluctuates from 1 to 30.

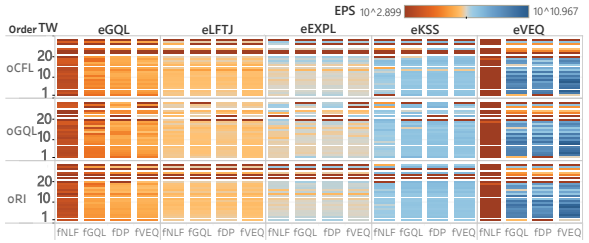


Fig. 19. Varying the treewidth (shorted as TW).

As the treewidth increases, there is a significant downward trend in EPS, indicating the rising of query difficulty. When the treewidth is intermediate, eVEQ tends to outperform other engine methods. However, at small treewidths, the efficiency of eVEQ noticeably deteriorates. This decline may be attributed to the simplistic structure of tree-like queries, where the application of sophisticated pruning techniques might lead to suboptimal performance. Interestingly, the order is observed to influence the algorithm's performance, but not consistently. Under conditions of exceptionally small tree-widths, where cardinality estimation is assumed to be straightforward, data-dependent order (e.g., oGQL and oCFL) does not consistently outperform data-independent order (oRI). The phenomenon leads to minimal performance differences, deserving further investigation. In addition, the kernel and shell decomposition in eKSS depends on the given order, yet its performance is not heavily influenced.

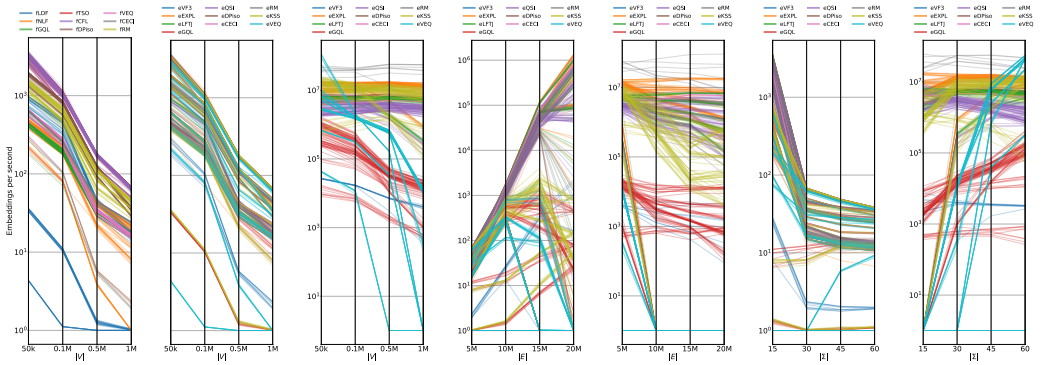
4.12 Evaluation of Scalability

In this subsection, we evaluate the scalability of the technical combinations. To ensure the consistency of graph features, we conduct experiments on synthetic datasets by varying vertex number $|V|$, edge number $|E|$, and vertex label size $|\Sigma|$. A collection of meaningful experimental results is shown in Figure 20.

4.12.1 Vary $|V|$. As shown in Figures 20(a) and 20(b), a decline in performance is observed across all algorithm combinations, with the increase of $|V|$ in ER graphs. The determinant factor of algorithmic performance lies predominantly in filtering techniques rather than enumerating techniques. Since ER graphs do not exhibit power-law properties, it is less likely to discover a large number of embeddings under the search subtree rooted in a single high-degree node. This leads to limited opportunities to enumerate multiple embeddings at a time or pruning during enumeration. Ranked by performance, the filtering techniques are fCFL, fDPiso, fCECI, fLDF, and fNLF, affirming the effectiveness of Filter Rule 3.1 for ER graphs.

For RMAT graphs, the importance of enumeration techniques is more evident. According to Figure 20(c), the optimal method is RM, followed by eEXPLORE and eKSS. The performance of these algorithm combinations remains relatively stable as the RMAT graph size increases. Meanwhile, the performance of eVEQ rapidly declines with an increase in the number of vertices. In contrast, eRM and eKSS once again demonstrate their performance stability.

4.12.2 Vary $|E|$. The performance of all algorithms in terms of EPS is shown in Figure 20(d) and Figure 20(e). For the native backtracking search, as the number of edges increases, with increased $|E|$, the success rate of backtrackings also rises, leading to an increase in performance measured by EPS. For eVEQ and eGQL, the increase in $|E|$ causes additional overhead, resulting in a performance curve that initially rises but then declines.



(a) Vary $|V|$ in ER, by filter. (b) Vary $|V|$ in ER, by enumeration. (c) Vary $|V|$ in RMAT, by enumeration. (d) Vary $|E|$ in ER, by enumeration. (e) Vary $|E|$ in RMAT, by enumeration. (f) Vary $|\Sigma|$ in ER, by enumeration. (g) Vary $|\Sigma|$ in RMAT, by enumeration.

Fig. 20. Evaluations of scalability.

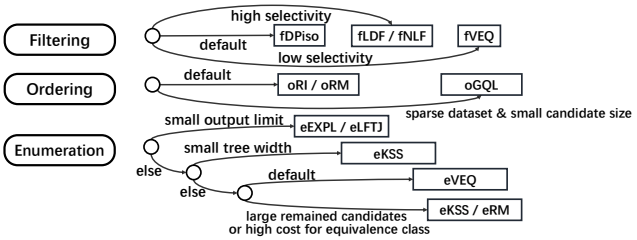


Fig. 21. Recommendation on algorithm selection.

For RMAT graphs, Figure 20(e) (by varying $|E|$) and Figure 20(c) (by varying $|V|$) exhibit a comparable trend. The performance of VEQ and GQL deteriorates substantially faster with the growth in the number of edges, compared to the other methods.

4.12.3 Vary $|\Sigma|$. On ER graphs, illustrated by Figure 20(f), all methods show a consistent decline in performance as $|\Sigma|$ increases. Generally, it is assumed that as the number of labels increases (leading to higher selectivity), the speed for the algorithm also increases correspondingly. However, since these results are still randomly scattered on ER graphs, leading to an actual increase in the time required to return a single embedding.

On RMAT graphs, Figure 20(f) is also similar to Figure 20(b). Regarding enumerating methods, RM method remains the best, followed by eEXPL and eKSS, and then eLFTJ and eQSI. With the growth in label size, eGQL and eVEQ exhibit rapid growth, especially eVEQ, which becomes the best method at $|\Sigma| = 60$. The growth of eGQL is attributed to fewer edge checkings needed when computing local candidates, while the higher selectivity of vertices provides eVEQ with more dynamic equivalence classes.

5 RECOMMENDATION AND DISCUSSION

5.1 Recommendation

Based on the experimental results, the recommendations for algorithm selection are as follows.

1) For filtering techniques, the fundamental principle is to make selections based on selectivity. The fDPiso method could serve as the default. For graphs with a rich number of labels and higher selectivity, fLDF or fNLF can be employed. Conversely, for graphs with fewer labels, or in cases where a large number of failing candidates are detected during enumeration, the fVEQ method might be worth considering.

2) For ordering techniques, we can take oRI or oRM as the default choice but turn to oGQL when the dataset is sparse and the candidate sizes are small.

3) For the enumeration process, techniques like eEXPL or eLFTJ can be employed when the output limit is relatively small. For query graphs with small treewidth, eKSS serves as an appropriate choice. In most other situations, eVEQ is a good choice. However, if an excessive number of candidates persist, or if the computational cost of eVEQ's equivalence class becomes substantial during execution, it may be advantageous to use eKSS or eRM.

5.2 Discussion

We conduct a comprehensive evaluation and analysis of the methods and find current trends focused on improving the backtracking framework. We can classify these methods into two types. One type such as RM and KSS enumerates multiple embeddings at a time while the other type like VEQ and GuP performs pruning during enumeration. We believe that enhancing the backtracking framework remains a promising direction for future progress. Going forward, challenges are how to integrate the above two types of methods, and to mitigate the suboptimal worst-case performance of pruning during enumeration methods.

We find that the choice of limiting reported embeddings has a considerable effect on the rankings of these methods, suggesting that the results of experimental evaluations in some past studies may have been biased. Hence, we propose a novel metric *embeddings per second* and re-evaluate all the algorithms. It has been commonly assumed that the use of pruning in enumeration techniques is beneficial mainly for large and complex queries [74]. Our empirical results show that such pruning techniques may be also efficient for simple queries, particularly when there are a large number of embeddings.

By exploring the entire algorithm design space, our research reveals that the interactions between various techniques play a crucial role. For example, KSS relies on the ordering method employed. We compare individual techniques when combined in their optimal combinations, to minimize potential interaction effects. In addition, we observe that combining different techniques could yield better performance and the ideal combination varies across these datasets. For future algorithm design, it's better to focus on the interplay between techniques, beyond optimizing individual ones.

6 CONCLUSIONS

In this paper, we conduct a comprehensive survey and experimental study of subgraph matching, focusing on three key issues: identifying the current trend, ensuring unbiasedness, and investigating the potential interactions. We identify that current trends focus on enhancing backtracking searches, whose promising advantages have also been confirmed in our experiments. Beyond fixing the output limit, we unbiasedly evaluate the performance of the algorithms by using an effective metric EPS. To study the interaction effect between individual techniques, we evaluate and analyze the original algorithms, as well as feasible combinations, through both real-world and synthetic graphs.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (No. U23A20496), Shanghai Science and Technology Innovation Action Plan (No. 21511100401), and GuangDong Basic and Applied Basic Research Foundation (No. 2019B1515120048).

REFERENCES

[1] Christopher R. Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun, and Christopher Ré. 2017. EmptyHeaded: A Relational Engine for Graph Processing. *ACM Trans. Database Syst.* 42, 4, Article 20 (oct 2017), 44 pages. https://doi.org/10.1145/3129246

[2] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-Coding. *J. ACM* 42, 4 (jul 1995), 844–856. https://doi.org/10.1145/210332.210337

[3] Khaled Ammar, Frank McSherry, Semih Salihoglu, and Manas Joglekar. 2018. Distributed evaluation of subgraph queries using worstcase optimal lowmemory dataflows. *Proceedings of the VLDB Endowment* (2018).

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

[5] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. 2015. Design and Implementation of the LogicBlox System. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 1371–1382. https://doi.org/10.1145/2723372.2742796

[6] Bibek Bhattarai, Hang Liu, and H. Howie Huang. 2019. CECI: Compact Embedding Cluster Index for Scalable Subgraph Matching. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1447–1462.

[7] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient Subgraph Matching by Postponing Cartesian Products. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 1199–1214. https://doi.org/10.1145/2882903.2915236

[8] Vincenzo Bonnici, Alfredo Ferro, Rosalba Giugno, Alfredo Pulvirenti, and Dennis Shasha. 2010. Enhancing Graph Database Indexing by Suffix Tree Structure. In *Proceedings of the 5th IAPR International Conference on Pattern Recognition in Bioinformatics* (Nijmegen, The Netherlands) *(PRIB'10)*. Springer-Verlag, Berlin, Heidelberg, 195–203.

[9] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. 2013. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics* 14, 7 (22 Apr 2013), S13. https://doi.org/10.1186/1471-2105-14-S7-S13

[10] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-Relational Data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (Lake Tahoe, Nevada) *(NIPS'13)*. Curran Associates Inc., Red Hook, NY, USA, 2787–2795.

[11] Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. 2018. Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2018), 804–818. https://doi.org/10.1109/TPAMI.2017.2696940

[12] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. *R-MAT: A Recursive Model for Graph Mining*. Society for Industrial and Applied Mathematics, 442–446. https://doi.org/10.1137/1.9781611972740.43 0.

[13] Peter Pin-Shan Chen. 1976. The Entity-Relationship Model—toward a Unified View of Data. *ACM Trans. Database Syst.* 1, 1 (mar 1976), 9–36. https://doi.org/10.1145/320434.320440

[14] Xuhao Chen, Roshan Dathathri, Gurbinder Gill, Loc Hoang, and Keshav Pingali. 2021. Sandslash: A Two-Level Framework for Efficient Graph Pattern Mining. In *Proceedings of the ACM International Conference on Supercomputing* (Virtual Event, USA) *(ICS '21)*. Association for Computing Machinery, New York, NY, USA, 378–391. https://doi.org/10.1145/3447818.3460359

[15] Xuhao Chen, Roshan Dathathri, Gurbinder Gill, and Keshav Pingali. 2020. Pangolin: An Efficient and Flexible Graph Mining System on CPU and GPU. *Proc. VLDB Endow.* 13, 8 (apr 2020), 1190–1205. https://doi.org/10.14778/3389133.3389137

[16] Xuhao Chen, Tianhao Huang, Shuotao Xu, Thomas Bourgeat, Chanwoo Chung, and Arvind Arvind. 2021. FlexMiner: A Pattern-Aware Accelerator for Graph Pattern Mining. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 581–594. https://doi.org/10.1109/ISCA52012.2021.00052

[17] Raffaele Di Natale, Alfredo Ferro, Rosalba Giugno, Misael Mongiovì, Alfredo Pulvirenti, and Dennis Shasha. 2010. SING: Subgraph search In Non-homogeneous Graphs. *BMC Bioinformatics* 11, 1 (19 Feb 2010), 96. https://doi.org/10.1186/1471-2105-11-96

[18] Vinicius Dias, Carlos H. C. Teixeira, Dorgival Guedes, Wagner Meira, and Srinivasan Parthasarathy. 2019. Fractal: A General-Purpose Graph Pattern Mining System. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1357–1374. https://doi.org/10.1145/3299869.3319875

[19] Wenfei Fan. 2012. Graph Pattern Matching Revised for Social Network Analysis. In *Proceedings of the 15th International Conference on Database Theory* (Berlin, Germany) *(ICDT '12)*. Association for Computing Machinery, New York, NY,

USA, 8–21. https://doi.org/10.1145/2274576.2274578

[20] Wenfei Fan, Tao He, Longbin Lai, Xue Li, Yong Li, Zhao Li, Zhengping Qian, Chao Tian, Lei Wang, Jingbo Xu, Youyang Yao, Qiang Yin, Wenyuan Yu, Jingren Zhou, Diwen Zhu, and Rong Zhu. 2021. GraphScope: A Unified Engine for Big Graph Processing. *Proc. VLDB Endow.* 14, 12 (jul 2021), 2879–2892. https://doi.org/10.14778/3476311.3476369

[21] Milton Friedman. 1937. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *J. Amer. Statist. Assoc.* 32, 200 (1937), 675–701. https://doi.org/10.1080/01621459.1937.10503522 arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1937.10503522

[22] Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., USA.

[23] Rosalba Giugno, Vincenzo Bonnici, Nicola Bombieri, Alfredo Pulvirenti, Alfredo Ferro, and Dennis Shasha. 2013. Grapes: A software for parallel searching on biological graphs targeting multi-core architectures. *PloS one* 8, 10 (2013), e76911.

[24] Wentian Guo, Yuchen Li, and Kian-Lee Tan. 2022. Exploiting Reuse for GPU Subgraph Enumeration. *IEEE Transactions on Knowledge and Data Engineering* 34, 9 (2022), 4231–4244. https://doi.org/10.1109/TKDE.2020.3035564

[25] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook Shin Han. 2019. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *SIGMOD 2019 - Proceedings of the 2019 International Conference on Management of Data.* 1429–1446.

[26] 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. https://doi.org/10.1145/2463676.2465300

[27] W. K. Hastings. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57, 1 (04 1970), 97–109. https://doi.org/10.1093/biomet/57.1.97 arXiv:https://academic.oup.com/biomet/article-pdf/57/1/97/23940249/57-1-97.pdf

[28] Huahai He and Ambuj K. Singh. 2008. Graphs-at-a-Time: Query Language and Access Methods for Graph Databases. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (Vancouver, Canada) *(SIGMOD '08).* Association for Computing Machinery, New York, NY, USA, 405–418. https://doi.org/10.1145/1376616.1376660

[29] Maarten Houbraken, Sofie Demeyer, Tom Michoel, Pieter Audenaert, Didier Colle, and Mario Pickavet. 2014. The Index-Based Subgraph Matching Algorithm with General Symmetries (ISMAGS): Exploiting Symmetry for Faster Subgraph Enumeration. *PLOS ONE* 9, 5 (05 2014), 1–15. https://doi.org/10.1371/journal.pone.0097896

[30] Kasra Jamshidi, Rakesh Mahadasa, and Keval Vora. 2020. Peregrine: A Pattern-Aware Graph Mining System. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20).* Association for Computing Machinery, New York, NY, USA, Article 13, 16 pages. https://doi.org/10.1145/3342195.3387548

[31] Tatiana Jin, Boyang Li, Yichao Li, Qihui Zhou, Qianli Ma, Yunjian Zhao, Hongzhi Chen, and James Cheng. 2023. Circinus: Fast Redundancy-Reduced Subgraph Matching. *Proc. ACM Manag. Data* 1, 1, Article 12 (may 2023), 26 pages. https://doi.org/10.1145/3588692

[32] Xin Jin and Longbin Lai. 2019. MPMatch: A Multi-core Parallel Subgraph Matching Algorithm. In *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW).* 241–248. https://doi.org/10.1109/ICDEW.2019.000-6

[33] Xin Jin, Zhengyi Yang, Xuemin Lin, Shiyu Yang, Lu Qin, and You Peng. 2021. FAST: FPGA-based Subgraph Matching on Massive Graphs. In *2021 IEEE 37th International Conference on Data Engineering (ICDE).* 1452–1463. https://doi.org/10.1109/ICDE51399.2021.00129

[34] Alpár Jüttner and Péter Madarasi. 2018. VF2++—An improved subgraph isomorphism algorithm. *Discrete Applied Mathematics* 242 (2018), 69–81. https://doi.org/10.1016/j.dam.2018.02.018 Computational Advances in Combinatorial Optimization.

[35] Alpár Jüttner and Péter Madarasi. 2018. VF2++—An improved subgraph isomorphism algorithm. *Discrete Applied Mathematics* 242 (2018), 69–81. https://doi.org/10.1016/j.dam.2018.02.018 Computational Advances in Combinatorial Optimization.

[36] Chathura Kankanamge, Siddhartha Sahu, Amine Mhedbhi, Jeremy Chen, and Semih Salihoglu. 2017. Graphflow: An Active Graph Database. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17).* Association for Computing Machinery, New York, NY, USA, 1695–1698. https://doi.org/10.1145/3035918.3056445

[37] Farzad Khorasani, Rajiv Gupta, and Laxmi N. Bhuyan. 2015. Scalable SIMD-Efficient Graph Processing on GPUs. In *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques (PACT '15).* 39–50.

[38] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong, and Wook-Shin Han. 2021. Versatile Equivalences: Speeding up Subgraph Query Processing and Subgraph Matching. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21).* Association for Computing Machinery, New

York, NY, USA, 925–937. https://doi.org/10.1145/3448016.3457265

[39] Hyeonji Kim, Juneyoung Lee, Sourav S. Bhowmick, Wook-Shin Han, JeongHoon Lee, Seongyun Ko, and Moath H.A. Jarrah. 2016. DUALSIM: Parallel Subgraph Enumeration in a Massive Graph on a Single Machine. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 1231–1245. https://doi.org/10.1145/2882903.2915209

[40] Jinha Kim, Hyungyu Shin, Wook-Shin Han, Sungpack Hong, and Hassan Chafi. 2015. Taming Subgraph Isomorphism for RDF Query Processing. *Proc. VLDB Endow.* 8, 11 (jul 2015), 1238–1249. https://doi.org/10.14778/2809974.2809985

[41] Kyoungmin Kim, In Seo, Wook-Shin Han, Jeong-Hoon Lee, Sungpack Hong, Hassan Chafi, Hyungyu Shin, and Geonhwa Jeong. 2018. TurboFlux: A Fast Continuous Subgraph Matching System for Streaming Graph Data. In *Proceedings of the 2018 International Conference on Management of Data*. 411–426.

[42] Raphael Kimmig, Henning Meyerhenke, and Darren Strash. 2017. Shared Memory Parallel Subgraph Enumeration. *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2017), 519–529.

[43] Karsten Klein, Nils Kriege, and Petra Mutzel. 2011. CT-Index: Fingerprint-Based Graph Indexing Combining Cycles and Trees. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering (ICDE '11)*. IEEE Computer Society, USA, 1115–1126. https://doi.org/10.1109/ICDE.2011.5767909

[44] Longbin Lai, Lu Qin, Xuemin Lin, and Lijun Chang. 2015. Scalable subgraph enumeration in mapreduce. *Proceedings of the VLDB Endowment* 8, 10 (2015), 974–985.

[45] Longbin Lai, Lu Qin, Xuemin Lin, Ying Zhang, Lijun Chang, and Shiyu Yang. 2016. Scalable distributed subgraph enumeration. *Proceedings of the VLDB Endowment* 10, 3 (2016), 217–228.

[46] Longbin Lai, Zhu Qing, Zhengyi Yang, Xin Jin, Zhengmin Lai, Ran Wang, Kongzhang Hao, Xuemin Lin, Lu Qin, Wenjie Zhang, et al. 2019. Distributed subgraph matching on timely dataflow. *Proceedings of the VLDB Endowment* 12, 10 (2019), 1099–1112.

[47] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. 2012. An In-Depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. *Proc. VLDB Endow.* 6, 2 (dec 2012), 133–144. https://doi.org/10.14778/2535568.2448946

[48] Guanfeng Li, Li Yan, and Zongmin Ma. 2019. An approach for approximate subgraph matching in fuzzy RDF graph. *Fuzzy Sets and Systems* 376 (2019), 106–126. https://doi.org/10.1016/j.fss.2019.02.021 Theme: Computer Science.

[49] TingHuai Ma, Siyang Yu, Jie Cao, Yuan Tian, Abdullah Al-Dhelaan, and Mznah Al-Rodhaan. 2018. A Comparative Study of Subgraph Matching Isomorphic Methods in Social Networks. *IEEE Access* 6 (2018), 66621–66631. https://doi.org/10.1109/ACCESS.2018.2875262

[50] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A System for Large-Scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, Indiana, USA) *(SIGMOD '10)*. Association for Computing Machinery, New York, NY, USA, 135–146. https://doi.org/10.1145/1807167.1807184

[51] Daniel Mawhirter and Bo Wu. 2019. AutoMine: Harmonizing High-Level Abstraction and High Performance for Graph Mining. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 509–523. https://doi.org/10.1145/3341301.3359633

[52] Ciaran McCreesh, Patrick Prosser, Christine Solnon, and James Trimble. 2018. When Subgraph Isomorphism is Really Hard, and Why This Matters for Graph Databases. *J. Artif. Int. Res.* 61, 1 (jan 2018), 723–759.

[53] Robert Ryan McCune, Tim Weninger, and Greg Madey. 2015. Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing. *ACM Comput. Surv.* 48, 2, Article 25 (oct 2015), 39 pages. https://doi.org/10.1145/2818185

[54] Amine Mhedhbi and Semih Salihoglu. 2019. Optimizing Subgraph Queries by Combining Binary and Worst-Case Optimal Joins. *Proc. VLDB Endow.* 12, 11 (jul 2019), 1692–1704. https://doi.org/10.14778/3342263.3342643

[55] Seunghwan Min, Sung Gwan Park, Kunsoo Park, Dora Giammarresi, Giuseppe F. Italiano, and Wook-Shin Han. 2021. Symmetric Continuous Subgraph Matching with Bidirectional Dynamic Programming. *Proc. VLDB Endow.* 14, 8 (2021), 1298–1310.

[56] Peter Bjorn Nemenyi. 1963. *Distribution-free multiple comparisons.* Princeton University.

[57] 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. https://doi.org/10.1145/3196959.3196990

[58] Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2012. Worst-case optimal join algorithms. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. 37–48.

[59] H. Q. Ngo, C Ré, and A. Rudra. 2014. Skew Strikes Back: New Developments in the Theory of Join Algorithms. *Acm Sigmod Record* 42, 4 (2014), 5–16.

[60] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and Complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3, Article 16 (sep 2009), 45 pages. https://doi.org/10.1145/1567274.1567278

[61] Miao Qiao, Hao Zhang, and Hong Cheng. 2017. Subgraph matching: on compression and computation. *Proceedings of the VLDB Endowment* 11, 2 (2017), 176–188.

[62] Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. Real-Time Constrained Cycle Detection in Large Dynamic Graphs. *Proc. VLDB Endow.* 11, 12 (aug 2018), 1876–1888. https://doi.org/10.14778/3229863.3229874

[63] Abdul Quamar, Amol Deshpande, and Jimmy Lin. 2016. NScale: Neighborhood-Centric Large-Scale Graph Analytics in the Cloud. *The VLDB Journal* 25, 2 (apr 2016), 125–150. https://doi.org/10.1007/s00778-015-0405-2

[64] Pedro Ribeiro, Pedro Paredes, Miguel E. P. Silva, David Aparicio, and Fernando Silva. 2021. A Survey on Subgraph Counting: Concepts, Algorithms, and Applications to Network Motifs and Graphlets. *ACM Comput. Surv.* 54, 2, Article 28 (mar 2021), 36 pages. https://doi.org/10.1145/3433652

[65] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. 2017. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *Proc. VLDB Endow.* 11, 4 (dec 2017), 420–431.

[66] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. 2018. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *Proc. VLDB Endow.* 11, 4 (oct 2018), 420–431. https://doi.org/10.1145/3164135.3164139

[67] Ahmet Erdem Sariyuce, C. Seshadhri, Ali Pinar, and Umit V. Catalyurek. 2015. Finding the Hierarchy of Dense Subgraphs Using Nucleus Decompositions. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 927–937. https://doi.org/10.1145/2736277.2741640

[68] C. Seshadhri. 2023. Some Vignettes on Subgraph Counting Using Graph Orientations. In *26th International Conference on Database Theory (ICDT 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 255)*, Floris Geerts and Brecht Vandevoort (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 3:1–3:10. https://doi.org/10.4230/LIPIcs.ICDT.2023.3

[69] Yingxia Shao, Bin Cui, Lei Chen, Lin Ma, Junjie Yao, and Ning Xu. 2014. Parallel subgraph listing in a large-scale graph. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of Data*. 625–636.

[70] Tianhui Shi, Mingshu Zhai, Yi Xu, and Jidong Zhai. 2020. GraphPi: High Performance Graph Pattern Matching through Effective Redundancy Elimination. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 100, 14 pages.

[71] Richard M. Stallman and Gerald J. Sussman. 1976. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artif. Intell.* 9 (1976), 135–196.

[72] Shixuan Sun, Yulin Che, Lipeng Wang, and Qiong Luo. 2019. Efficient Parallel Subgraph Enumeration on a Single Machine. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 232–243. https://doi.org/10.1109/ICDE.2019.00029

[73] Shixuan Sun and Qiong Luo. 2019. Scaling up subgraph query processing with efficient subgraph matching. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 220–231.

[74] Shixuan Sun and Qiong Luo. 2020. In-Memory Subgraph Matching: An In-Depth Study. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1083–1098. https://doi.org/10.1145/3318464.3380581

[75] 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. https://doi.org/10.14778/3425879.3425888

[76] Shixuan Sun, Xibo Sun, Bingsheng He, and Qiong Luo. 2022. RapidFlow: An Efficient Approach to Continuous Subgraph Matching. *Proc. VLDB Endow.* 15, 11 (jul 2022), 2415–2427. https://doi.org/10.14778/3551793.3551803

[77] Xibo Sun and Qiong Luo. 2023. Efficient GPU-Accelerated Subgraph Matching. *Proc. ACM Manag. Data* 1, 2, Article 181 (jun 2023), 26 pages. https://doi.org/10.1145/3589326

[78] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient subgraph matching on billion node graphs. *Proceedings of the VLDB Endowment* (2012).

[79] Carlos H. C. Teixeira, Alexandre J. Fonseca, Marco Serafini, Georgos Siganos, Mohammed J. Zaki, and Ashraf Aboulnaga. 2015. Arabesque: A System for Distributed Graph Mining. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 425–440. https://doi.org/10.1145/2815400.2815410

[80] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference*. Springer, 210–218.

[81] J. R. Ullmann. 1976. An Algorithm for Subgraph Isomorphism. *J. ACM* 23, 1 (jan 1976), 31–42. https://doi.org/10.1145/321921.321925

[82] T. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *ICDT*.

[83] Todd L Veldhuizen. 2012. Leapfrog triejoin: a worst-case optimal join algorithm. *arXiv preprint arXiv:1210.0481* (2012).

[84] Carletti Vincenzo, Pasquale Foggia, Pierluigi Ritrovato, Mario Vento, and Vincenzo Vigilante. 2019. *A Parallel Algorithm for Subgraph Isomorphism.* 141–151. https://doi.org/10.1007/978-3-030-20081-7_14

[85] Kai Wang, Zhiqiang Zuo, John Thorpe, Tien Quang Nguyen, and Guoqing Harry Xu. 2018. RStream: Marrying Relational Algebra with Streaming for Efficient Graph Mining on a Single Machine. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 763–782.

[86] Zhaokang Wang, Rong Gu, Weiwei Hu, Chunfeng Yuan, and Yihua Huang. 2019. BENU: Distributed subgraph enumeration with backtracking-based framework. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 136–147.

[87] Zhaokang Wang, Weiwei Hu, Guowang Chen, Chunfeng Yuan, Rong Gu, and Yihua Huang. 2021. Towards Efficient Distributed Subgraph Enumeration Via Backtracking-Based Framework. *IEEE Transactions on Parallel and Distributed Systems* 32, 12 (2021), 2953–2969. https://doi.org/10.1109/TPDS.2021.3076246

[88] Lizhi Xiang, Arif Khan, Edoardo Serra, Mahantesh Halappanavar, and Aravind Sukumaran-Rajam. 2021. CuTS: Scaling Subgraph Isomorphism on Distributed Multi-GPU Systems Using Trie Based Data Structure. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 69, 14 pages. https://doi.org/10.1145/3458817.3476214

[89] Su Xunbin, Lin Yinnian, and Lei Zou. 2023. FASI: FPGA-friendly Subgraph Isomorphism on Massive Graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*.

[90] Da Yan, Yingyi Bu, Yuanyuan Tian, and Amol Deshpande. 2017. Big graph analytics platforms. *Foundations and Trends in Databases* 7, 1-2 (2017), 1–195.

[91] Da Yan, Yingyi Bu, Yuanyuan Tian, Amol Deshpande, and James Cheng. 2016. Big Graph Analytics Systems. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 2241–2243. https://doi.org/10.1145/2882903.2912566

[92] Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. 2014. *Pregel+: A Distributed Graph Computing Framework with Effective Message Reduction.* The Chinese University of Hong Kong, Hong Kong, China. http://www.cse.cuhk.edu.hk/pregelplus/

[93] Da Yan, Guimu Guo, Md Mashiur Rahman Chowdhury, M. Tamer Özsu, Wei-Shinn Ku, and John C. S. Lui. 2020. G-thinker: A Distributed Framework for Mining Subgraphs in a Big Graph. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 1369–1380. https://doi.org/10.1109/ICDE48307.2020.00122

[94] Xifeng Yan, Philip S. Yu, and Jiawei Han. 2004. Graph Indexing: A Frequent Structure-Based Approach. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (Paris, France) *(SIGMOD '04)*. Association for Computing Machinery, New York, NY, USA, 335–346. https://doi.org/10.1145/1007568.1007607

[95] 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. https://doi.org/10.1145/3588695

[96] Zhengyi Yang, Longbin Lai, Xuemin Lin, Kongzhang Hao, and Wenjie Zhang. 2021. HUGE: An Efficient and Scalable Subgraph Enumeration System. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2049–2062. https://doi.org/10.1145/3448016.3457237

[97] Li Zeng, Lei Zou, M. Tamer Özsu, Lin Hu, and Fan Zhang. 2020. GSI: GPU-friendly Subgraph Isomorphism. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 1249–1260. https://doi.org/10.1109/ICDE48307.2020.00112

[98] Yuejia Zhang, Weiguo Zheng, Zhijie Zhang, Peng Peng, and Xuecang Zhang. 2022. Hybrid Subgraph Matching Framework Powered by Sketch Tree for Distributed Systems. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1031–1043. https://doi.org/10.1109/ICDE53745.2022.00082

[99] Yuejia Zhang, Weiguo Zheng, Zhijie Zhang, Peng Peng, and Xuecang Zhang. 2022. Hybrid Subgraph Matching Framework Powered by Sketch Tree for Distributed Systems. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1031–1043. https://doi.org/10.1109/ICDE53745.2022.00082

[100] Cheng Zhao, Zhibin Zhang, Peng Xu, Tianqi Zheng, and Jiafeng Guo. 2020. Kaleido: An Efficient Out-of-core Graph Mining System on A Single Machine. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 673–684. https://doi.org/10.1109/ICDE48307.2020.00064

[101] Peixiang Zhao, Jeffrey Xu Yu, and Philip S. Yu. 2007. Graph Indexing: Tree + Delta >= Graph. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (Vienna, Austria) *(VLDB '07)*. VLDB Endowment, 938–949.

Received 15 July 2023; revised 20 October 2023; accepted 20 November 2023