

Fast Maximum Common Subgraph Search: A Redundancy-Reduced Backtracking Approach

KAIQIANG YU, Nanyang Technological University, Singapore

KAIXIN WANG, Beijing University of Technology, China

CHENG LONG*, Nanyang Technological University, Singapore

LAKS LAKSHMANAN, The University of British Columbia, Canada

REYNOLD CHENG, The University of Hong Kong, China

Given two input graphs, finding the largest subgraph that occurs in both, i.e., finding the maximum common subgraph, is a fundamental operator for evaluating the similarity between two graphs in graph data analysis. Existing works for solving the problem are of either theoretical or practical interest, but not both. Specifically, the algorithms with a theoretical guarantee on the running time are known to be not practically efficient; algorithms following the recently proposed backtracking framework called McSplit, run fast in practice but do not have any theoretical guarantees. In this paper, we propose a new backtracking algorithm called RRSplit, which at once achieves better practical efficiency and provides a non-trivial theoretical guarantee on the worst-case running time. To achieve the former, we develop a series of reductions and upper bounds for reducing redundant computations, i.e., the time for exploring some unpromising branches of exploration that hold no maximum common subgraph. To achieve the latter, we formally prove that RRSplit incurs a worst-case time complexity which matches the best-known complexity for the problem. Finally, we conduct extensive experiments on several benchmark graph collections, and the results demonstrate that our algorithm outperforms the practical state-of-the-art by several orders of magnitude.

CCS Concepts: • **Mathematics of computing** → **Graph algorithms**; • **Information systems** → **Information retrieval query processing**.

Additional Key Words and Phrases: Graph similarity search; maximum common subgraph; subgraph matching; subgraph isomorphism

ACM Reference Format:

Kaiqiang Yu, Kaixin Wang, Cheng Long, Laks Lakshmanan, and Reynold Cheng. 2025. Fast Maximum Common Subgraph Search: A Redundancy-Reduced Backtracking Approach. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 160 (June 2025), 27 pages. <https://doi.org/10.1145/3725404>

1 Introduction

Graphs have been increasingly adopted to capture the relationships among entities in various domains, including social media, biological networks, communication networks, collaboration networks etc. As a result, graph data analysis has gained great attention in the recent years. One of the most fundamental problems in graph analysis is *maximum common subgraph search*, which is widely used to measure the similarity of two graphs [4, 13, 30–34, 39, 49, 50, 54, 55]. More precisely,

*Corresponding author (c.long@ntu.edu.sg)

Authors' Contact Information: Kaiqiang Yu, Nanyang Technological University, Singapore, kaiqiang002@e.ntu.edu.sg; Kaixin Wang, Beijing University of Technology, Beijing, China, kaixin001@e.ntu.edu.sg; Cheng Long, Nanyang Technological University, Singapore, c.long@ntu.edu.sg; Laks Lakshmanan, The University of British Columbia, Vancouver, Canada, laks@cs.ubc.ca; Reynold Cheng, The University of Hong Kong, Hong Kong, China, ckcheng@cs.hku.hk.



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

© 2025 Copyright held by the owner/author(s).

ACM 2836-6573/2025/6-ART160

<https://doi.org/10.1145/3725404>

a common subgraph between two graphs Q and G refers to a subgraph that appears in both Q and G . Conceptually, it is defined by a pair of subgraphs, $q = (V_q, E_q)$ of Q and $g = (V_g, E_g)$ of G , and a bijection mapping $\phi : V_q \rightarrow V_g$ such that q and g are *isomorphic* under ϕ , which we denote by $\langle q, g, \phi \rangle$. Given two graphs Q and G , the problem asks for the common subgraph $\langle q, g, \phi \rangle$ with the maximum number of vertices in q (equiv. g).

The maximum common subgraph search problem has many applications across various disciplines, and has been widely studied [1, 4, 13, 26, 28, 30–34, 39, 47, 49, 50, 54, 55]. To be specific, it offers an operator for evaluating the similarity between graphs in graph database systems [51] and thus has found a wide range of real applications, including cheminformatics [2, 40], communication networks [36], software analysis [37, 46], biochemistry [8, 15, 27], and image segmentation [19]. For example, the similarity between two molecules is calculated based on the maximum common subgraph between them [15]. Therefore, in drug discovery and analysis, it is used to quickly identify a small group of compounds with similar substructures (which tend to preserve similar properties) for further analysis, so as to reduce the manual labor and shorten the cycle of discovery [15]. Besides, the maximum common subgraph search problem generalizes the well-studied *subgraph matching* problem [3, 6, 7, 14, 17, 18, 21, 24, 41, 43–45, 48]. More specifically, given a data graph G and a query graph Q , subgraph matching seeks to find if there is an embedding¹ from Q to G , where an embedding means a 1-1 function from the nodes of Q to those of G which preserves edges, i.e., the embedding is a witness that Q is isomorphic to a subgraph of G . Embedding or subgraph matching is a strong requirement. In real applications data quality is a real concern [5, 12] and thus an embedding from Q to G may fail to exist, with subgraph matching yielding no results. In such circumstances, finding a maximum common subgraph is a graceful relaxation of the problem which may still yield useful results. If the largest common subgraph is very similar to Q , it can serve as an approximation to the embedding of Q . Given this, in this paper, we focus on finding the maximum common subgraph between two given graphs.

Challenges and existing methods. Detecting the maximum common subgraph is quite challenging as noted in the literature. Specifically, it is NP-hard [29] and is shown to be at least as hard to approximate as the maximum clique problem: it does not admit any r -approximate algorithm that runs in polynomial time (unless $P = NP$), for any $r \geq 1$ [22]. Existing works for solving the problem are of either theoretical or practical interest. On the one hand, some algorithms are designed to improve theoretical time complexity [1, 26, 28, 47]. Assume that Q has the number of vertices no larger than G , i.e., $|V_Q| \leq |V_G|$. They have gradually improved the worst-case time complexity from $O^*(1.19^{|V_Q||V_G|})$ [28] to $O^*(|V_Q|^{(|V_G|+1)})$ [26], and to $O^*((|V_Q|+1)^{|V_G|})$ [47], which is the best-known worst-case time complexity for the problem. Here, O^* suppresses polynomials. However, these algorithms are of theoretical interest only and not efficient in practice. This is mainly because their theoretical results rely on some sophisticated data structures while maintaining them introduces a huge amount of time and/or memory overhead in practice, e.g., the intermediate graph structure built from Q and G has $|V_Q| \times |V_G|$ vertices and could be very large if G is large (e.g., with million nodes). On the other hand, quite a few algorithms have been developed towards improving the practical performance [28, 30–34, 49, 55]. They are all backtracking (also known as branch-and-bound) algorithms, among which the recent works [30, 31, 55] are based on a newly proposed backtracking framework called McSplit [33]. McSplit recursively partitions the search space (i.e., the set of possible common subgraphs) to multiple sub-spaces via a process of branching. Each sub-space corresponds to a branch. Furthermore, McSplit uses the upper bound on the size of common subgraphs that could be found within a branch for reducing redundant computations associated with exploring those branches that do not lead to finding the maximum

¹Not to be confused with graph embeddings.

common subgraph. Specifically, it prunes those branches that have upper bounds no larger than the size of the largest common subgraph seen so far. However, *these algorithms provide no non-trivial theoretical guarantee on the worst-case time complexity.*

We note that maximum common subgraph is closely related to a well-known graph similarity measure, namely graph edit distance (GED) [9]. The GED between two graphs Q and G is the cost of the least-cost edit path, i.e., a sequence of edit operations that transform Q to G . Under a special cost function which does not charge for edge insertion/deletion and charges an infinite cost for node/edge substitution, the GED computation has been shown to be equivalent to finding the maximum common subgraph [9]. However, recent state-of-the-art approaches [10, 11, 16, 25, 38] for computing GED assume different cost functions where the above equivalence does not hold. Specifically, these cost functions do not ignore edge deletion/insertion costs and usually assign the same positive finite costs to various edit operators. Consequently, state of the art algorithms for GED cannot lead to efficient solutions for computing the maximum common subgraph.

Our method. In this paper, we develop an efficient backtracking algorithm called `RRSplit`, which leverages newly-designed reductions and new upper bounds for decreasing the redundant computations. With $|V_Q| \leq |V_G|$, `RRSplit` achieves a worst-case time complexity of $O^*((|V_G| + 1)^{|V_Q|})$, matching the best-known worst-case time complexity for the problem [47]. We note that this theoretical result is remarkable since (1) the algorithm with the best-known time complexity in [47] is of theoretical interest only and is not practically efficient and (2) the algorithms with the best-known practical performance, i.e., `McSplit` [33] and its variants, do not have any theoretical guarantee on the worst-case time complexity. Specifically, `RRSplit` combines the following two kinds of reductions: (i) vertex-equivalence-based and (ii) maximality-based, and a vertex-equivalence-based upper bound that we establish.

Vertex-equivalence-based reductions reduce the redundant computations induced by *common subgraph isomorphism (cs-isomorphism)*. Given two common subgraphs $\langle q, g, \phi \rangle$ and $\langle q', g', \phi' \rangle$ of graphs G and Q , they are said to be *common subgraph isomorphic (cs-isomorphic for short)* if and only if q is isomorphic to q' (or equivalently, g is isomorphic to g'). All cs-isomorphic common subgraphs evidently share the same structural information, and exploring all of them is clearly redundant. To reduce this redundancy, we take two sufficient conditions into consideration when deciding whether we can prune a branch. Specifically, for any common subgraph $\langle q, g, \phi \rangle$ to be found in a branch, if there exists another that is cs-isomorphic to $\langle q, g, \phi \rangle$ (Condition 1) and has been found before (Condition 2), we can safely prune the branch. To facilitate the reduction, we will leverage the *vertex equivalence* property [35] and an *auxiliary data structure* for verifying Condition 1 and Condition 2, respectively (details in Section 4.1).

Maximality-based reductions capture the redundant computations induced by *maximality*. Specifically, a common subgraph $\langle q, g, \phi \rangle$ is maximal if and only if there does not exist any other common subgraph $\langle q', g', \phi' \rangle$ such that q and g are subgraphs of q' and g' , respectively. Therefore, the maximum common subgraph is a maximal common subgraph, and those branches that hold only non-maximal ones would incur redundant computations. To reduce them, we observe one necessary condition for a branch to hold the largest common subgraphs (details in Section 4.2). As a result, we can safely prune those branches that violate the condition.

Furthermore, we leverage the vertex equivalence property to derive a new vertex-equivalence-based upper bound, which is tighter than the existing one [33] and thus can help to prune more branches (details in Section 4.3).

Contributions. We make the following contributions.

- We introduce vertex-equivalence-based reductions for reducing the redundant computation induced by cs-isomorphism (Section 4.1). We further propose maximality-based reductions for

pruning those branches that hold non-maximal common subgraphs only (Section 4.2). Finally, we develop a new vertex-equivalence-based upper bound for pruning more branches (Section 4.3).

- We propose a new backtracking algorithm called **RRSplit**, which is based on the newly-designed reductions. It has a worst-case time complexity that matches the best-known time complexity (of the algorithms that are of theoretical interest only) (Section 4.4).
- We conduct an extensive empirical evaluation on several benchmark graph collections. Our experiments reveal that our algorithm **RRSplit** runs several orders of magnitude faster than the state-of-the-art **McSplitDAL** on the majority of the tested input instances (Section 5).

Section 2 provides a formal statement of the problem studied. Section 3 reviews the existing framework **McSplit** and its state-of-the-art variant **McSplitDAL**. In Section 6 we discuss related work and conclude the paper in Section 7. Due to space limits, we omit some proofs and sketch others, and all proofs can be found in the technical report [53].

2 Preliminaries

In this paper, we focus on undirected and unweighted simple graphs without self-loops and parallel edges. For ease of presentation, we focus on graphs without vertex labels, but our methods can be easily adapted to vertex-labeled graphs. Consider two graphs $Q = (V_Q, E_Q)$ and $G = (V_G, E_G)$. For simplicity, we let u and v (and their primed or index variants) denote a vertex in Q and G respectively. Given a vertex set $X \subseteq V_Q$, we use $Q[X]$ to denote the subgraph of Q induced by X , i.e., $Q[X] = (X, \{(u, u') \in E_Q \mid u, u' \in X\})$. All subgraphs in this paper are induced subgraphs. We let $q = (V_q, E_q)$ denote an arbitrary induced subgraph of Q . Given $u \in V_Q$, we denote by $N(u, V_Q)$ (resp. $\overline{N}(u, V_Q)$) the set of neighbours (resp. non-neighbours) of u in $Q[V_Q]$. We use a similar notation for neighbours and non-neighbours of vertices in G .

We review the definition of graph isomorphism for simple graphs without labels.

Definition 1 (Graph isomorphism [34]). Q is said to be isomorphic to G if and only if there exists a **bijection** $\phi : V_Q \rightarrow V_G$ such that

$$\forall u, u' \in V_Q : (u, u') \in E_Q \iff (\phi(u), \phi(u')) \in E_G. \quad (1)$$

Note that two isomorphic graphs are structurally equivalent, and thus we have $|V_Q| = |V_G|$ and $|E_Q| = |E_G|$. We next review induced subgraph isomorphism for graphs.

Definition 2 (Induced subgraph isomorphism [34]). Q is said to be (induced) subgraph isomorphic to G if and only if there exists an **injection** $\phi : V_Q \rightarrow V_G$ such that

$$\forall u, u' \in V_Q : (u, u') \in E_Q \iff (\phi(u), \phi(u')) \in E_G. \quad (2)$$

Notice that induced subgraph isomorphism is a special case of graph isomorphism. The injection mapping $\phi : V_Q \rightarrow V_G$ is also known as *embedding* of Q into G , and thus we have $|V_Q| \leq |V_G|$ and $|E_Q| \leq |E_G|$. The subgraph matching problem aims to find all embeddings of a small query graph Q in a large data graph G . The common induced subgraph is defined as follows.

Definition 3 (Common induced subgraph [34]). A common subgraph of Q and G , denoted by $\langle q, g, \phi \rangle$, is defined as a triple consisting of an induced subgraph q of Q , an induced subgraph g of G , and a bijection $\phi : V_q \rightarrow V_g$, such that q is isomorphic to g under ϕ .

By the size of a common subgraph $\langle q, g, \phi \rangle$ we mean the number of vertices in q or g . Clearly, the size of a common subgraph is at most $\min\{|V_Q|, |V_G|\}$. Sometimes, for ease of presentation, we represent a common subgraph $\langle q, g, \phi \rangle$ by a set of vertex pairs $\{\langle u, \phi(u) \rangle \mid u \in V_q\}$.

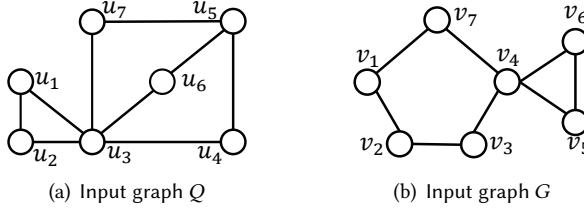


Fig. 1. Input graphs used throughout the paper

EXAMPLE 1. Consider the input graphs in Figure 1. The graphs $q := Q[u_1, u_2, u_3, u_4, u_7]$ and $g := G[v_3, v_4, v_5, v_6, v_7]$ form a common subgraph with size 5 under the bijection $\phi := \{u_1 \rightarrow v_5, u_2 \rightarrow v_6, u_3 \rightarrow v_4, u_4 \rightarrow v_3, u_7 \rightarrow v_7\}$.

We are ready to formulate the problem studied in this paper.

PROBLEM 1 (MAXIMUM COMMON SUBGRAPH [29]). Given two graphs Q and G , the Maximum Common Subgraph (MCS) problem aims to find the maximum common subgraph of Q and G , i.e., a common subgraph with the largest number of vertices.

Note that the MCS problem is a generalization of subgraph matching: there is a MCS between Q and G whose size is $|Q|$ iff Q is isomorphic to a subgraph of G . It is well known that MCS is NP-hard [29] and is hard to approximate, i.e., there is no r -approximate PTIME algorithm for the problem for any $r \geq 1$ [22], unless $P=NP$.

3 The Basic Framework: McSplit

Overview. McSplit, a *backtracking* (aka *branch-and-bound*) algorithm, has been widely adopted for the MCS problem in recent years and has achieved the state-of-the-art performance in practice [4, 30, 31, 55]. The key idea of McSplit is to recursively expand a partial solution S (which is the largest common subgraph seen so far) via a process of *branching*. Specifically, the branching process partitions the current problem instance of finding the maximum common subgraph into several subproblem instances. Each problem instance corresponds to a *branch* and is denoted by (S, C) . Here, S is a *partial solution* (i.e., set of vertex pairs) and C is the *candidate set* consisting of *candidate pairs* (i.e., $\langle u, v \rangle$) used to expand the partial solution S . Solving the instance or branch (S, C) means finding the largest common subgraph S^* in the branch; a common subgraph is said to be in a branch (S, C) if and only if it contains S and is within the set $S \cup C$, i.e., $S \subseteq S^* \subseteq S \cup C$. Given two vertex subsets $V_q \subseteq V_Q$ and $V_g \subseteq V_G$, we consider all pairs of vertices from V_q and V_g , i.e., $V_q \times V_g = \{\langle u, v \rangle \mid u \in V_q, v \in V_g\}$. Note that solving the initial branch $(\emptyset, V_Q \times V_G)$ finds the largest common subgraph of Q and G .

To solve a branch (S, C) , the branching process selects a vertex u appearing in C as a *branching vertex*, and then creates two groups of new sub-branches by either including u into the solution or discarding u from the candidate set and thus also from the solution. Specifically, **in the first group**, each formed branch includes into S one candidate pair containing u and excludes from C all pairs containing u (note that a common subgraph has each vertex appearing in at most one pair); consequently, for each candidate pair that contains u , i.e., $\langle u, v \rangle$, we form a new branch corresponding to $(S \cup \{\langle u, v \rangle\}, C \setminus \langle u, v \rangle)$, where $C \setminus \langle u, v \rangle$ denotes the set obtained by removing from C all candidate pairs containing u or v , formally,

$$C \setminus \langle u, v \rangle := C \setminus ((\{u\} \times V_g) \cup (V_q \times \{v\})). \quad (3)$$

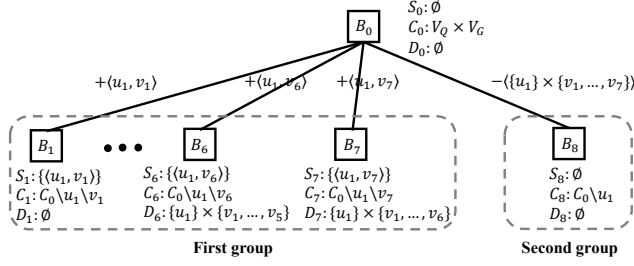


Fig. 2. Illustrating the backtracking process (“+” means to move vertex pairs from C to S and “-” means to remove vertex pairs from C)

In the second group, we form only one branch by excluding from C all candidate pairs containing u , i.e., $(S, C \setminus u)$. Clearly, the solution to (S, C) is the largest one among those found from the branches above. We illustrate this next.

EXAMPLE 2. Consider the given pair of input graphs in Figure 1. The splitting process is partially depicted in Figure 2 (ignore the “D” terms in the figure for now). For the initial branch $B_0 = (\emptyset, \{u_1, u_2, \dots, u_7\} \times \{v_1, v_2, \dots, v_7\})$, McSplit selects the branching vertex u_1 , and then creates the first group of branches $B_i = (\{ \langle u_1, v_i \rangle \}, \{u_2, \dots, u_7\} \times (\{v_1, v_2, \dots, v_7\} \setminus \{v_i\}))$ for $1 \leq i \leq 7$, each of which includes one candidate pair $\langle u_1, v_i \rangle$ into the solution, and the second group of a single branch, namely $B_8 = (\emptyset, \{u_2, \dots, u_7\} \times \{v_1, v_2, \dots, v_7\})$, which excludes u_1 from the solution.

To improve the efficiency, McSplit further applies a *reduction rule* and an *upper-bound-based pruning rule* for a newly formed branch (S, C) . Specifically, the reduction rule removes from the candidate set C those candidate pairs $\langle u, v \rangle$ that cannot form a common subgraph with S , i.e., $S \cup \{ \langle u, v \rangle \}$ cannot be a common subgraph, thus narrowing the search space: note that any supergraph of a non-common subgraph cannot be a common subgraph and thus we can remove them safely. The upper-bound-based pruning rule computes an upper bound on the size of the largest common subgraph in the branch and prunes the branch if the upper bound is no larger than the size of the largest common subgraph found so far (details will be discussed in Section 4.3). Below, we formally state the reduction rule.

Reduction rule. Consider a branch (S, C) . A candidate pair $\langle u, v \rangle$ in C cannot form any common subgraph with S if there exists a vertex pair $\langle u', v' \rangle$ in S such that u and v are not simultaneously adjacent or non-adjacent to u' and v' , respectively.

The soundness of this rule immediately follows from Definition 3. We note that the above reduction rule can be applied in a recursive way and the refined candidate set C can be split as several subsets, i.e., $C = X_1 \times Y_1 \cup \dots \cup X_c \times Y_c$ where c is a positive integer and X_i and X_j (resp. Y_i and Y_j) are disjoint. We show this as follows. More precisely, starting from the basis case of the initial branch B_0 with $S_0 = \emptyset$ and $C_0 = V_Q \times V_G$, none of the candidate pairs in C_0 can be pruned by the reduction rule since S_0 is empty. Then, consider the recursive case of a branch $B = (S, C)$ with $C = X_1 \times Y_1 \cup \dots \cup X_c \times Y_c$ where c is a positive integer and X_i and X_j (resp. Y_i and Y_j) are disjoint for $1 \leq i \leq c$. For one sub-branch $(S \cup \{ \langle u, v \rangle \}, C \setminus u \setminus v)$ formed in the first group by including $\langle u, v \rangle$ to S , the candidate set $C \setminus u \setminus v$ can be refined as

$$\bigcup_{i=1}^c N(u, X_i) \times N(v, Y_i) \cup \overline{N}(u, X_i \setminus \{u\}) \times \overline{N}(v, Y_i \setminus \{v\}). \quad (4)$$

since those vertex pairs in $N(u, X_i) \times \overline{N}(v, Y_i \setminus \{v\}) \cup \overline{N}(u, X_i \setminus \{u\}) \times N(v, Y_i)$ can be pruned by the above reduction. Clearly, $N(u, X_i)$ and $\overline{N}(u, X_i \setminus \{u\})$ (resp. $N(v, Y_i)$ and $\overline{N}(v, Y_i \setminus \{v\})$) are disjoint

Algorithm 1: An existing framework: McSplit [33]

Input: Two graphs $Q = (V_Q, E_Q)$ and $G = (V_G, E_G)$
Output: The maximum common subgraph

```

1  $S^* \leftarrow \emptyset$ ; // Global variable
2 McSplit-Rec( $\emptyset, V_Q \times V_G$ ); Return  $S^*$ ;
3 Procedure McSplit-Rec( $S, C$ )
4   if  $|S| > |S^*|$  then  $S^* \leftarrow S$ ;                                */
   /* Termination
5   if  $C = \emptyset$  or the upper bound is no larger than  $|S^*|$  then return;  */
   /* Branching
6   Select a branching vertex  $u$  and a branching subset  $X \times Y$  from  $\mathcal{P}(C)$  based on a policy;
7    $Y_{temp} \leftarrow Y$ ;
   /* First group: branches formed by including  $u$                                 */
8   for  $i = 1, 2, \dots, |Y|$  do
9     Select and move a vertex  $v$  from  $Y_{temp}$  based on a policy;
10    Create a candidate set  $C_i$  based on  $\langle u, v \rangle$  and Equation (4);
11    McSplit-Rec( $S \cup \{\langle u, v \rangle\}, C_i$ );
   /* Second group: one branch formed by excluding  $u$                                 */
12  McSplit-Rec( $S, C \setminus u$ );

```

for $1 \leq i \leq c$. For one sub-branch $(S, C \setminus u)$ formed in the second group, none of the candidate pairs in $C \setminus u$ can be pruned by the reduction rule since S remains unchanged. Suppose $u \in X_i$, then the refined candidate set $C \setminus u$ can be represented by $X_1 \times Y_1 \cup \dots \cup (X_i \setminus \{u\}) \times Y_i \cup \dots \cup X_c \times Y_c$ where any two subsets are disjoint as well.

We remark that all candidate sets C mentioned in the following sections refer to the ones refined by the reduction rule and thus can be represented by $C = X_1 \times Y_1 \cup \dots \cup X_c \times Y_c$, for some c . Given this, we define

$$\mathcal{P}(C) = \{X_i \times Y_i \mid 1 \leq i \leq c\}. \quad (5)$$

EXAMPLE 3. Consider branch B_6 in the first group in Figure 2. We have $X_1 = N(u_1, V_Q) = \{u_2, u_3\}$, $X_2 = \overline{N}(u_1, V_Q \setminus \{u_1\}) = \{u_4, u_5, u_6, u_7\}$, $Y_1 = \{v_4, v_5\}$ and $Y_2 = \{v_1, v_2, v_3, v_7\}$. Therefore, the candidate set becomes $\{u_2, u_3\} \times \{v_4, v_5\} \cup \{u_4, u_5, u_6, u_7\} \times \{v_1, v_2, v_3, v_7\}$. Then, consider a sub-branch of B_6 formed by further including $\langle u_7, v_7 \rangle$. We can deduce the candidate set by splitting $X_1 \times Y_1$ to $\{u_3\} \times \{v_4\} \cup \{u_2\} \times \{v_5\}$ and splitting $X_2 \times Y_2$ to $\{u_5\} \times \{v_1\} \cup \{u_4, u_6\} \times \{v_2, v_3\}$.

Algorithm Outline. We summarize the details of McSplit in Algorithm 1. It maintains the currently found largest common subgraph S^* (Line 4) and terminates the branch by upper-bound-based pruning (Line 5). Besides, it branches by selecting (from $\mathcal{P}(C)$) a vertex u and the corresponding subset $X \times Y$, called *branching subset*, that u belongs to (Line 6, note that all candidate pairs containing u are within $X \times Y$), and creates two groups of branches as discussed before (Lines 8-12). For the first group, the ordering of formed branches depends on that of the candidate pairs to be included into S , which is specified by a policy (Line 9). We note that McSplit *adopts heuristic policies for selecting $X \times Y$, u , and v .*

Existing algorithms that are based on McSplit differ in the strategies of optimizing the policies of selecting vertices in line 6 and line 9 (e.g., via some learning-based techniques) to find the largest common subgraph as soon as possible during the recursion [30, 31, 55]. However, these algorithms

(1) provide no theoretical guarantee on the worst-case time complexity and (2) still suffer from efficiency issues in practice due to significant redundant computations.

4 Redundancy-Reduced Splitting: RRSplit

In this part, we present our backtracking algorithm called RRSplit. First, we propose a vertex-equivalence-based reduction for pruning those redundant branches that hold all common subgraphs cs-isomorphic to one that has been already found (Section 4.1). Second, we introduce a newly-designed maximality-based reduction for pruning those redundant branches that hold only non-maximal common subgraphs (Section 4.2). Third, we develop a new vertex-equivalence-based upper bound on the size of common subgraphs that can be found in a branch for further pruning those branches that hold only small common subgraphs (Section 4.3). Finally, we summarize the RRSplit algorithm, which is based on the above reductions, and analyze its worst-case time complexity (Section 4.4). In particular, we show RRSplit has a worst-case time complexity of $O^*((|V_G| + 1)^{|V_Q|})$, matching the best-known worst-case time complexity of the state of the art. We will later show (Section 5) that unlike the state of the art, RRSplit is very efficient in practice.

4.1 Vertex-Equivalence-based Reduction

We first introduce the concept of *common subgraph isomorphism* (cs-isomorphism).

Definition 4 (Common subgraph isomorphism). Consider two common subgraphs $\langle q, g, \phi \rangle$ and $\langle q', g', \phi' \rangle$ of graphs G and Q . They are said to be common subgraph isomorphic (cs-isomorphic) if and only if q is isomorphic to q' (or equiv., g is isomorphic to g').

All cs-isomorphic common subgraphs evidently share the same structural information, and exploring all of them is clearly redundant. We reduce this redundancy as follows. For a common subgraph $\langle q, g, \phi \rangle$ to be found in a branch, we can safely ignore the common subgraph $\langle q, g, \phi \rangle$, if there exists another one $\langle q', g', \phi' \rangle$ that satisfies the following two conditions:

- **Condition 1:** $\langle q', g', \phi' \rangle$ is cs-isomorphic to $\langle q, g, \phi \rangle$;
- **Condition 2:** $\langle q', g', \phi' \rangle$ has been found before.

To facilitate the reduction, we make use of Condition 1 and Condition 2, for which we will leverage the *vertex equivalence* property and an *auxiliary data structure* respectively.

Vertex equivalence. The *structural equivalence* property has been widely used to speed up subgraph matching tasks [23, 35, 52]. Conceptually, two vertices are *structurally equivalent* if and only if they have the *same* set of neighbours. Formally,

Definition 5 (Structural equivalence [35]). Two vertices u and v in Q are structurally equivalent, denoted $u \sim v$, if and only if

$$\forall u' \in V_Q, (u, u') \in E_Q \Leftrightarrow (v, u') \in E_Q. \quad (6)$$

Clearly, structural equivalence is an equivalence relation. Therefore, we can partition the vertices of graph Q into equivalence classes, with the equivalence class of vertex $u \in V_Q$ defined as

$$\Psi(u) := \{u' \in V_Q \mid u' \sim u\}, \quad (7)$$

where $u \in V_Q$ is a representative of class $\Psi(u)$. We remark that this process can be done in $O(|V_Q|\delta_Q d_Q)$ time where δ_Q and d_Q are the degeneracy and the maximum degree of the graph Q , respectively [23, 35, 52].

Based on vertex equivalence, we can identify several common subgraphs that are cs-isomorphic to a given one $\langle q, g, \phi \rangle$ by swapping one vertex in V_q with its structurally equivalent counterpart, which falls into two cases. In specific, consider a vertex u in V_q and one of its structurally equivalent

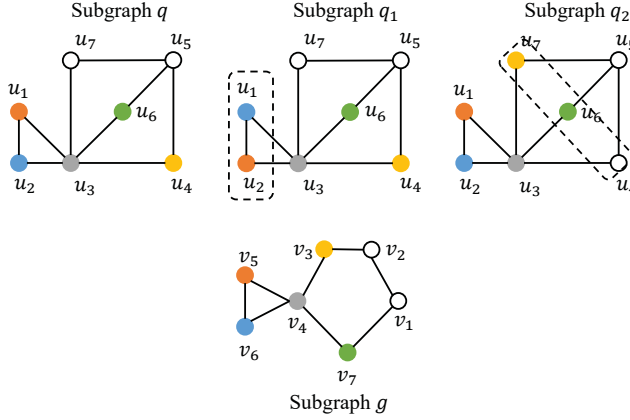


Fig. 3. Illustrating cs-isomorphism and vertex equivalence (vertices, denoted by colored bullet circles, induce subgraphs q , q_1 , q_2 and g ; vertices in $\{u_1, u_2\}$ and $\{u_4, u_6, u_7\}$ are structurally equivalent, respectively; $\langle q, g, \phi \rangle$ is cs-isomorphic to $\langle q_1, g, \phi_1 \rangle$ (Case 1, say, exchange the mapped vertices of u_1 and u_2) and $\langle q_2, g, \phi_2 \rangle$ (Case 2, say, replace u_4 with u_7) where vertices with the same color indicate the bijection)

counterparts u_{equ} in $\Psi(u)$. We can obtain a cs-isomorphic common subgraph in two cases. If u_{equ} is also in V_q , we can exchange the mapped vertices of u_{equ} and u , i.e., we replace $\langle u, \phi(u) \rangle$ and $\langle u_{equ}, \phi(u_{equ}) \rangle$ with $\langle u, \phi(u_{equ}) \rangle$ and $\langle u_{equ}, \phi(u) \rangle$; Otherwise, we replace u with u_{equ} , i.e., replacing $\langle u, \phi(u) \rangle$ with $\langle u_{equ}, \phi(u) \rangle$. Formally, we have the following lemma, which can be easily verified (see the examples in Figure 3 for a visual illustration of the lemma).

LEMMA 1. *Let $S = \langle q, g, \phi \rangle$ be a common subgraph of given graphs Q and G , u be a vertex in V_q and u' be a vertex in $\Psi(u)$. Then one of the following cases holds.*

Case 1: $u' \in V_q$. $S' = S \setminus \{\langle u, \phi(u) \rangle, \langle u', \phi(u') \rangle\} \cup \{\langle u, \phi(u') \rangle, \langle u', \phi(u) \rangle\}$ is a common subgraph cs-isomorphic to S .

Case 2: $u' \notin V_q$. $S' = S \setminus \{\langle u, \phi(u) \rangle\} \cup \{\langle u', \phi(u) \rangle\}$ is a common subgraph cs-isomorphic to S .

Auxiliary data structure. To facilitate the verification of Condition 2, i.e., whether a common subgraph that is cs-isomorphic to a current one has been found before, we introduce a new data structure, namely exclusion set (denoted by D). D is recursively maintained for each branch, and thus each branch is denoted by (S, C, D) . Specifically, D is a set of vertex pairs that have been considered for expanding the partial solution and must not be included in any common subgraphs within the branch. Formally, the exclusion set is maintained as follows (illustrated in Figure 2 – see the “ D ” terms now!).

- **Initialization.** The exclusion set is initialized to be empty at the initial branch, i.e., $(\emptyset, V_Q \times V_G, \emptyset)$.
- **Recursive update.** Consider the branching at a branch (S, C, D) . For the first group where the i^{th} sub-branch (S_i, C_i, D_i) is formed by including $\langle u, v_i \rangle$ into S , we update the exclusion set to $D_i = D \cup \{\langle u, v_1 \rangle, \langle u, v_2 \rangle, \dots, \langle u, v_{i-1} \rangle\}$. For the second group where one sub-branch (S', C', D') is formed, we set $D' = D$.

Consider a branch (S, C, D) and a vertex pair $\langle u', v' \rangle$ in the exclusion set D , as shown in Figure 4. There exists an ancestor of (S, C, D) , denoted by $(S_{anc}, C_{anc}, D_{anc})$, where u' is selected as the branching vertex. Clearly, $\langle u', v' \rangle$ is not in D_{anc} and will be added to D_{anc} after B'_{anc} is formed,

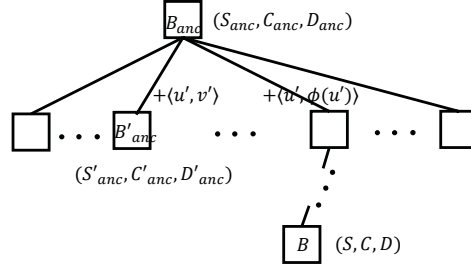


Fig. 4. Illustrating the exclusion set D ($\langle u', v' \rangle$ is a vertex pair in D ; B_{anc} is an ancestor of B , where u' is selected as the branching vertex)

i.e., more precisely $D'_{anc} = D_{anc} \cup \{\langle u', v' \rangle\}$. Therefore, all common subgraphs within the sub-branch B'_{anc} , which must contain $\langle u', v' \rangle$, have been found before (S, C, D) . This will help us verify Condition 2.

Based on vertex equivalence and exclusion set, we are now ready to develop the reductions. Consider the branching process at a branch $(S = \langle q, g, \phi \rangle, C, D)$ where $X \times Y$ in $\mathcal{P}(C)$ and vertex u in X are selected as the branching subset and branching vertex, respectively.

Reduction at the first group. Consider a sub-branch formed at the first group by including one vertex pair $\langle u, v \rangle$ where $v \in Y$. We note that *each* common subgraph S_{sub} to be found in this sub-branch must include $\langle u, v \rangle$. We observe that if *there exists a vertex pair* $\langle u_{equ}, v \rangle$ in D such that u_{equ} is structurally equivalent to u , i.e., $u_{equ} \in \Psi(u)$, Conditions 1 & 2 hold for S_{sub} and thus the branch can be pruned. Below, we elaborate on the details.

We first show that Condition 1 holds: Clearly, S contains a vertex pair $\langle u_{equ}, \phi(u_{equ}) \rangle$ since otherwise D will not include $\langle u_{equ}, v \rangle$ according to the maintenance of D . Therefore, we can construct the following common subgraph S_{iso} , which is cs-isomorphic to S_{sub} based on Case 1 of Lemma 1 (essentially, we exchange the mapped vertices of u_{equ} and u).

$$S_{iso} = S_{sub} \setminus \{\langle u, v \rangle, \langle u_{equ}, \phi(u_{equ}) \rangle\} \cup \{\langle u_{equ}, v \rangle, \langle u, \phi(u_{equ}) \rangle\} \quad (8)$$

Clearly, S_{iso} is cs-isomorphic to S_{sub} given that u and u_{equ} are structurally equivalent.

We next show that S_{iso} has been found before and thus Condition 2 holds: In specific, consider an ancestor of (S, C, D) , denoted by $(S_{anc}, C_{anc}, D_{anc})$, where u_{equ} is selected as the branching vertex, as visually illustrated in Figure 4. Since S_{iso} contains $\langle u_{equ}, v \rangle$, we check whether it has been found in the sub-branch of $(S_{anc}, C_{anc}, D_{anc})$, which is formed by including $\langle u_{equ}, v \rangle$. The answer is interestingly positive. The rationale behind is that (1) $S_{iso} \setminus \{\langle u, \phi(u_{equ}) \rangle\}$ is a subset of $S_{anc} \cup C_{anc}$ (this can be easily verified by the facts that $S_{sub} \subseteq S_{anc} \cup C_{anc}$ and $\langle u_{equ}, v \rangle \in C_{anc}$) and (2) $\langle u, \phi(u_{equ}) \rangle$ constructed by ours is in C_{anc} (this holds due to the vertex equivalence between u and u_{equ}). Therefore, S_{iso} is a common subgraph in $(S_{anc}, C_{anc}, D_{anc})$, i.e., $S_{anc} \subseteq S_{iso} \subseteq S_{anc} \cup C_{anc}$, and has been found before (S, C, D) since it contains $\langle u_{equ}, v \rangle$.

In summary, we have the following lemma and reduction rule.

LEMMA 2. *Let (S, C, D) be a branch. Common subgraph S_{iso} defined in Equation (8) has been found before the formation of (S, C, D) .*

PROOF. We note that the recursive branching process forms a recursion tree where each tree node corresponds to a branch. Consider the path from the initial branch $(\emptyset, V_Q \times V_G, \emptyset)$ to (S, C, D) , there exists an ascendant branch of (S, C, D) , denoted by $B_{asc} = (S_{asc}, C_{asc}, D_{asc})$, where u_{equ} is selected as the branching vertex, since $\langle u_{equ}, \phi(u_{equ}) \rangle$ is in S . We can see that there exists one sub-branch $B'_{asc} = (S'_{asc}, C'_{asc}, D'_{asc})$ of B_{asc} formed by including $\langle u_{equ}, v \rangle$, and all common subgraphs within

B'_{asc} have been found before the formation of (S, C, D) , since $\langle u_{equ}, v \rangle$ is in D . We then show that common subgraph S_{iso} can be found within B'_{asc} , i.e., $S'_{asc} \subseteq S_{iso} \subseteq S'_{asc} \cup C'_{asc}$. First, we have $S'_{asc} \subseteq S_{iso}$ since (1) $S'_{asc} = S_{asc} \cup \{\langle u_{equ}, v \rangle\}$, (2) S_{sub} is a common subgraph in B_{asc} and thus $S_{asc} \subseteq S_{sub}$, (3) S_{asc} does not include $\langle u_{equ}, \phi(u_{equ}) \rangle$ or $\langle u, v \rangle$ since they are in C_{asc} and will be included to the partial solution at B'_{asc} and $(S \cup \{\langle u, v \rangle\}, C \setminus u \setminus v)$, and thus (4) by combining all the above, we have $S'_{asc} = S_{asc} \cup \{\langle u_{equ}, v \rangle\} \subseteq S_{sub} \cup \{\langle u_{equ}, v \rangle\} \setminus \{\langle u_{equ}, \phi(u_{equ}) \rangle, \langle u, v \rangle\} \subseteq S_{iso}$. Second, we have $S_{iso} \subseteq S'_{asc} \cup C'_{asc}$ based on the following two facts.

- **Fact 1.** $S_{sub} \setminus \{\langle u_{equ}, \phi(u_{equ}) \rangle, \langle u, v \rangle\} \subseteq S'_{asc} \cup C'_{asc}$.
- **Fact 2.** $\langle u_{equ}, v \rangle \in S'_{asc}$ and $\langle u, \phi(u_{equ}) \rangle \in C'_{asc}$.

The correctness of the above facts can be verified accordingly, for which we put the details in the technical report [53]. $\square$

Vertex-Equivalence-based reduction at the first group. Let $B = (S, C, D)$ be a branch. For a sub-branch of B formed by including a candidate pair $\langle u, v \rangle$ in the first group, it can be pruned if there exists a vertex pair $\langle u', v \rangle$ in D such that $u' \in \Psi(u)$.

EXAMPLE 4. Consider again the branching process at branch $B_6 = (S_6, C_6, D_6)$ in Figure 2. Suppose u_2 is selected as the branching vertex. Then, we can see that u_1 is in $\Psi(u_2)$ and $D_6 = \{u_1\} \times \{v_1, v_2, v_3, v_4, v_5\}$. Recall that $C_6 = \{u_2, u_3\} \times \{v_4, v_5\} \cup \{u_4, u_5, u_6, u_7\} \times \{v_1, v_2, v_3, v_7\}$. Thus, B_6 has two sub-branches which are formed by including $\langle u_2, v_4 \rangle$ or $\langle u_2, v_5 \rangle$, and they can be pruned based on the above reduction.

Reduction at the second group. Recall that vertex u is selected as the branching vertex. Consider the sub-branch formed by excluding u in the second group. We note that each common subgraph $S_{sub} = \langle q_{sub}, g_{sub}, \phi_{sub} \rangle$ to be found in this sub-branch must exclude vertex u . We observe that if S_{sub} contains a vertex u_{equ} , which is in $C \setminus u$ and is structurally equivalent to u , Condition 1&2 holds for S_{sub} .

In specific, Condition 1 holds since we can construct the following common subgraph which is cs-isomorphic to S_{sub} based on Case 2 of Lemma 1 (essentially, we replace u_{equ} with u).

$$S_{iso} = S_{sub} \setminus \{\langle u_{equ}, \phi_{sub}(u_{equ}) \rangle\} \cup \{\langle u, \phi_{sub}(u_{equ}) \rangle\} \quad (9)$$

Besides, we note that S_{iso} contains $\langle u, \phi_{sub}(u_{equ}) \rangle$ and common subgraphs found in the first group must include u . We thus check whether S_{iso} has been found in one sub-branch formed in the first group. The answer is also positive. The rationale behind is that (1) $\langle u, \phi_{sub}(u_{equ}) \rangle$ constructed by ours exists in C (this holds due to the vertex equivalence between u and u_{equ}) and (2) thus there exists a sub-branch formed by including $\langle u, \phi_{sub}(u_{equ}) \rangle$ in the first group, where S_{iso} has been found since it includes $\langle u, \phi_{sub}(u_{equ}) \rangle$.

In summary, we have the following lemma and reduction rule.

LEMMA 3. Let (S, C, D) be a branch where u is selected as the branching vertex. Common subgraph S_{iso} defined in Equation (9) has been found before the formation of $(S, C \setminus u, D)$ at the second group.

PROOF. First, we note that $\langle u_{equ}, \phi_{iso}(u_{equ}) \rangle$ is in $C \setminus u$ and also in C since otherwise S_{sub} cannot include $\langle u_{equ}, \phi_{iso}(u_{equ}) \rangle$. This is because (1) $S \subseteq S_{sub} \subseteq S \cup C \setminus u$ since S_{sub} is a common subgraph in the sub-branch $(S, C \setminus u, D)$ and (2) S does not include $\langle u_{equ}, \phi_{iso}(u_{equ}) \rangle$ since u_{equ} appears in $C \setminus u$. Second, we note that $\phi_{iso}(u_{equ})$ is in Y . Recall that $X \times Y$ is the branching set at (S, C, D) . This is because (1) u_{equ} is in the same subset X as u since u_{equ} and u are structurally equivalent and thus have the same set of neighbours and non-neighbours in q , and (2) $\langle u_{equ}, \phi_{iso}(u_{equ}) \rangle$ is in C as discussed before. Third, we can derive that there exists a sub-branch $(S \cup \{\langle u, \phi_{iso}(u_{equ}) \rangle\}, C \setminus u \setminus \phi_{iso}(u_{equ}), D')$, which is formed at branch (S, C, D) by including $\langle u, \phi_{iso}(u_{equ}) \rangle$ before the formation of $(S, C \setminus u, D)$, since $\phi_{iso}(u_{equ}) \in Y$. Forth, we show that S_{iso} is in $(S \cup \{\langle u, \phi_{iso}(u_{equ}) \rangle\}, C \setminus u \setminus \phi_{iso}(u_{equ}), D')$,

formally, $S \cup \{\langle u, \phi_{iso}(u_{equ}) \rangle\} \subseteq S_{iso} \subseteq S \cup \{\langle u, \phi_{iso}(u_{equ}) \rangle\} \cup (C \setminus u \setminus \phi_{iso}(u_{equ}))$. We have $S \subseteq S_{sub} \subseteq S \cup (C \setminus u)$ since S_{sub} is a common subgraph in $(S, C \setminus u, D)$. Let $S' = S \cup \{\langle u, \phi_{iso}(u_{equ}) \rangle\}$, it can be proved as below.

$$S \subseteq S_{sub} \subseteq S \cup (C \setminus u) \quad (10)$$

$$\Rightarrow S \subseteq S_{sub} \setminus \{\langle u_{equ}, \phi_{iso}(u_{equ}) \rangle\} \subseteq S \cup (C \setminus u \setminus \phi_{iso}(u_{equ})) \quad (11)$$

$$\Rightarrow S' \subseteq S_{iso} \subseteq S' \cup (C \setminus u \setminus \phi_{iso}(u_{equ})) \quad (12)$$

Note that Equation (11) holds since $\langle u_{equ}, \phi_{iso}(u_{equ}) \rangle$ is in S ; Equation (12) is derived by including the vertex pair $\langle u, \phi_{iso}(u_{equ}) \rangle$. $\square$

Vertex-Equivalence-based reduction at the second group. Let $B = (S, C, D)$ be a branch and $(S, C \setminus u, D)$ be the sub-branch formed in the second group by excluding all candidate pairs that consist of u . For a vertex u' appearing in $C \setminus u$, if u' is structurally equivalent to u , i.e., $u' \in \Psi(u)$, all candidate pairs that consist of u' can be pruned from $C \setminus u$.

EXAMPLE 5. Consider the branching process at B_0 where u_1 is the branching vertex in Figure 2. For sub-branch B_8 (which is the sub-branch at the second group), we can see that $\Psi(u_1) = \{u_1, u_2\}$ and thus the candidate set of B_8 can be reduced to $(V_Q \setminus \{u_1, u_2\}) \times V_G$, i.e., to $\{u_3, u_4, u_5, u_6, u_7\} \times \{v_1, v_2, \dots, v_7\}$.

4.2 Maximality-based Reduction

We introduce the redundancies induced by *non-maximality*. Clearly, a maximum common subgraph must be a maximal common subgraph. Therefore, exploring those branches that hold non-maximal common subgraphs only will incur redundant computations. Consider a current branch $B = (S, C, D)$. Note that there might exist multiple common subgraphs with the largest number of vertices in the branch. We observe that *there exists one largest common subgraph in B that must contain one specific candidate vertex pair $\langle u, v \rangle$* . Given this, we can remove this candidate vertex pair $\langle u, v \rangle$ from C to S , thereby producing one immediate sub-branch i.e., $(S \cup \{\langle u, v \rangle\}, C \setminus u \setminus v)$. Clearly, solving the resulting sub-branch is enough to find the largest common subgraph (since it holds all those common subgraphs that contain $\langle u, v \rangle$ in B). As a result, we can safely prune all other sub-branches for which the partial set and the candidate set do not include the candidate pair $\langle u, v \rangle$. Below, we elaborate on the details.

To be specific, we observe that there exists one largest common subgraph, denoted by S_{opt} , in B such that S_{opt} must contain a candidate vertex pair $\langle u, v \rangle$ if for any subset $X \times Y$ in $\mathcal{P}(C)$, u and v are simultaneously adjacent or non-adjacent to all other vertices in X and Y , respectively, i.e.,

$$\begin{aligned} \forall X \times Y \in \mathcal{P}(C) : (N(u, X) = X \setminus \{u\} \text{ and } N(v, Y) = Y \setminus \{v\}) \text{ or} \\ (N(u, X) = \emptyset \text{ and } N(v, Y) = \emptyset), \end{aligned} \quad (13)$$

Formally, we have the following lemma.

LEMMA 4. Let $B = (S, C, D)$ be a branch and $\langle u, v \rangle$ be a candidate vertex pair that satisfies the condition in Equation (13). There exists one largest common subgraph S_{opt} in the branch B such that S_{opt} contains $\langle u, v \rangle$.

PROOF. This can be proved by construction. Let $S^* = (q^*, g^*, \phi^*)$ be one largest common subgraph to be found in B . Note that if S^* contains the candidate vertex pair $\langle u, v \rangle$, we can finish the proof by constructing S_{opt} as S^* . Otherwise, if $\langle u, v \rangle$ is not in S^* , we prove the correctness by constructing one largest common subgraph S_{opt} to be found in B that contains candidate vertex pair $\langle u, v \rangle$, i.e., $S \subseteq S_{opt} \subseteq S \cup C$, $|S_{opt}| = |S^*|$ and $\langle u, v \rangle \in S_{opt}$. In general, there are four different cases, and the details can be found in the technical report [53]. $\square$

Consider a branch $B = (S, C, D)$ where $X^* \times Y^*$ in $\mathcal{P}(C)$ and u^* in X^* are selected as the branching subset and the branching vertex, respectively. Assume that there exists a vertex v in Y^* such that $\langle u^*, v \rangle$ satisfies the condition in Equation (13). Based on the above lemma, there exists one largest common subgraph in the branch B that contains candidate vertex pair $\langle u^*, v \rangle$. Therefore, we only need to form one sub-branch $(S \cup \{\langle u^*, v \rangle\}, C \setminus u^* \setminus v, D \cup \{u^*\} \times (Y^* \setminus \{v\}))$ since other formed sub-branches will exclude the candidate vertex $\langle u^*, v \rangle$ from the found common subgraphs. We note that the exclusion set of the formed sub-branch can be updated by $D \cup \{u^*\} \times (Y^* \setminus \{v\})$ to enhance the pruning power of the proposed reduction at the first group. In summary, we obtain the following reduction.

Maximality-based reduction. Let $B = (S, C, D)$ be a branch where $X \times Y$ in $\mathcal{P}(C)$ and u in X are selected as the branching subset and the branching vertex. If there exists a candidate vertex pair $\langle u, v \rangle$ in the candidate set such that $\langle u, v \rangle$ satisfies Equation (13), only one sub-branch $(S \cup \{\langle u, v \rangle\}, C \setminus u \setminus v, D \cup \{u\} \times (Y \setminus \{v\}))$ needs to be formed at B .

EXAMPLE 6. Consider the branching at branch $B_6 = (S_6, C_6, D_6)$ in Figure 2 where suppose u_2 is selected as the branching vertex. Recall that $C_6 = X_1 \times Y_1 \cup X_2 \times Y_2 = \{u_2, u_3\} \times \{v_4, v_5\} \cup \{u_4, u_5, u_6, u_7\} \times \{v_1, v_2, v_3, v_7\}$. We note that $\langle u_2, v_5 \rangle$ satisfies Equation (13) since (1) $N(u_2, X_1) = X_1 \setminus \{u_2\}$ and $N(v_5, Y_1) = Y_1 \setminus \{v_5\}$ and (2) $N(u_2, X_2) = \emptyset$ and $N(v_5, Y_2) = \emptyset$. Therefore, we only need to explore one sub-branch $(S_6 \cup \{\langle u_2, v_5 \rangle\}, C_6 \setminus u_2 \setminus v_5, D_6 \cup \{\langle u_2, v_4 \rangle\})$, and other two sub-branches formed at B_6 can be pruned.

4.3 Vertex-Equivalence-based Upper Bound

Consider a current branch (S, C, D) and the largest common subgraph S^* seen so far. Clearly, we can terminate the branch (S, C, D) , if the upper bound on the size of common subgraphs to be found in the branch (S, C, D) (or simply, the upper bound of (S, C, D)) is no larger than the size of S^* . The tighter the upper bound, the more branches we can prune.

Existing upper bound. Consider a common subgraph S_{sub} to be found in the branch (S, C, D) . For a subset $X \times Y$ in $\mathcal{P}(C)$, we can derive

$$|S_{sub} \cap X \times Y| \leq ub_{X,Y} := \min\{|X|, |Y|\} \quad (14)$$

since otherwise a common subgraph will contain two distinct vertex pairs $\langle u, v \rangle$ and $\langle u', v' \rangle$ such that $u = u'$ or $v = v'$ (which violates the definition of the bijection). Here, $ub_{X,Y}$ is the upper bound of the number of candidate pairs that are within $X \times Y$ and are in a common subgraph to be found in the branch (S, C, D) . Furthermore, since all subsets in $\mathcal{P}(C)$ are disjoint, the following existing upper bound of branch (S, C, D) , denoted by $ub_{S,C}$ [33], can be derived.

$$|S_{sub}| \leq ub_{S,C} := |S| + \sum_{X \times Y \in \mathcal{P}(C)} ub_{X,Y} \quad (15)$$

Motivation. We observe that the existing upper bound $ub_{X,Y}$ is not tight since some candidate vertex pairs in $X \times Y$ can be pruned from the candidate set C based on the proposed vertex-equivalence-based reductions. In specific, for a candidate vertex pair $\langle u, v \rangle$, if there exists a vertex pair $\langle u', v \rangle$ in D such that $u' \in \Psi(u)$, any common subgraph to be found within (S, C, D) cannot include $\langle u, v \rangle$ and thus $\langle u, v \rangle$ can be pruned from the candidate set C . Note that this can be easily verified based on the proposed reduction at the first group. Below, we introduce our upper bound derived with the aid of the structural equivalence on vertices.

New upper bound. Consider a subset $X \times Y$ in $\mathcal{P}(C)$. Let u be an arbitrary vertex in X . We partition X and Y as follows.

$$X_L = X \cap \Psi(u), X_R = X \setminus X_L \quad (16)$$

$$Y_L = \{v \mid \langle u', v \rangle \in D, u' \in \Psi(u)\}, Y_R = Y \setminus Y_L, \quad (17)$$

where X_L consists of those vertices in X that are structurally equivalent to u and Y_L consists of those vertices v in Y which appear in a vertex pair $\langle u', v \rangle$ in D where $u' \in \Psi(u)$. We then can partition $X \times Y$ as $X_L \times Y_L$, $X_L \times Y_R$, $X_R \times Y_L$ and $X_R \times Y_R$. Clearly, all vertex pairs in $X_L \times Y_L$ can be pruned as discussed before. We note that (1) S_{sub} contains at most $\min\{|X_R|, |Y|\}$ vertex pairs from $X_R \times Y_L$ and $X_R \times Y_R$ since otherwise there exists one vertex in $X_R \cup Y$ that appears in at least two distinct vertex pairs in S_{sub} and thus S_{sub} cannot be a common subgraph; and similarly (2) S_{sub} contains at most $\min\{|X_L|, |Y_R|, \max\{|Y| - |X_R|, 0\}\}$ vertex pairs from $X_L \times Y_R$ (note that the additional term $\max\{|Y| - |X_R|, 0\}$ is used to ensure that the sum of $\min\{|X_R|, |Y|\}$ and $\min\{|X_L|, |Y_R|, \max\{|Y| - |X_R|, 0\}\}$ is no larger than the existing upper bound $ub_{S,C}$). Therefore, S_{sub} contains at most $ub_{X,Y,D}$ vertex pairs from $X \times Y$, where

$$ub_{X,Y,D} := \min\{|X_R|, |Y|\} + \min\{|X_L|, |Y_R|, \max\{|Y| - |X_R|, 0\}\}. \quad (18)$$

Then, we can derive our upper bound of a branch (S, C, D) , denoted by $ub_{S,C,D}$, i.e.,

$$|S_{sub}| \leq ub_{S,C,D} := |S| + \sum_{X \times Y \in \mathcal{P}(C)} ub_{X,Y,D}. \quad (19)$$

In summary, we obtain our new upper bound $ub_{S,C,D}$ as above. It is not difficult to verify that our upper bound is tighter than the existing one, i.e., $ub_{S,C,D} \leq ub_{S,C}$: see Example 7 for an example where $ub_{S,C,D} < ub_{S,C}$.

LEMMA 5 (UPPER BOUND). *Let (S, C, D) be a branch. All common subgraphs to be found in (S, C, D) have the size at most $ub_{S,C,D}$.*

EXAMPLE 7. *Consider again the branching process at branch $B_6 = (S_6, C_6, D_6)$ in Figure 2. Recall that $C_6 = X_1 \times Y_1 \cup X_2 \times Y_2 = \{u_2, u_3\} \times \{v_4, v_5\} \cup \{u_4, u_5, u_6, u_7\} \times \{v_1, v_2, v_3, v_7\}$ and $D_6 = \{u_1\} \times \{v_1, v_2, \dots, v_5\}$. For $X_1 \times Y_1$, based on u_2 , we have $X_{1L} = \{u_2\}$, $X_{1R} = \{u_3\}$, $Y_{1L} = \{v_4, v_5\}$ and $Y_{1R} = \emptyset$. Thus, we have $ub_{X_1, Y_1, D_6} = \min\{1, 4\} + \min\{1, 0, \max\{1, 0\}\} = 1$. For $X_2 \times Y_2$, based on u_4 , we have $X_{2L} = \{u_4\}$, $X_{2R} = \{u_5, u_6, u_7\}$, $Y_{2L} = \emptyset$ and $Y_{2R} = \{v_1, v_2, v_3, v_7\}$. Thus, we have $ub_{X_2, Y_2, D_6} = \min\{3, 4\} + \min\{1, 4, \max\{1, 0\}\} = 4$. Therefore, we have $ub_{S_6, C_6, D_6} = 1 + 1 + 4 = 6$, which is smaller than the existing bound $ub_{S,C} = 7$.*

Remark. Our new upper bound $ub_{S,C,D}$ varies with different choices of u due to the partition of X and Y . We can potentially obtain a tighter upper bound by exploring all possible choices of u . However, it would introduce a large amount of time costs, thus degrading the performance of RRSplit. Therefore, as a trade-off, we randomly select u when computing the upper bound.

4.4 Summary and Analysis

Summary. We summarize our algorithm, namely RRSplit, in Algorithm 2, which incorporates the newly proposed vertex-equivalence-based reductions, the maximality-based reduction and the vertex-equivalence-based upper bound. Specifically, RRSplit differs with McSplit in the following aspects. (1) It maintains one additional auxiliary data structure, namely exclusion set D , for each formed branch, which is initialized as the empty set and recursively updated as discussed. (2) It prunes a branch (S, C, D) if the newly proposed vertex-equivalence-based upper bound $ub_{S,C,D}$ is no larger than the largest common subgraph size seen so far, i.e., $|S^*|$ (Line 7). We remark that $ub_{S,C,D}$ is tighter than the existing one $ub_{S,C}$, i.e., $ub_{S,C,D} \leq ub_{S,C}$ and thus more branches can be

pruned. (3) It creates only one sub-branch and prunes all others if the maximality-based reduction is triggered (Lines 9-11). (4) Based on the vertex-equivalence-based reduction, it prunes those sub-branches at the first group that hold all common subgraphs inside cs-isomorphic to the one found before (Lines 15-16), and refines the formed sub-branch at the second group by removing from the candidate set all those candidate vertex pairs consisting of a vertex in $\Psi(u)$ (Line 19). We remark that our implementation of **RRSplit** in the experiments adopts the same heuristic policies for selecting branching subset $X \times Y$, branching vertex u (Line 8) and vertex v (Line 14) as **McSplit** does. Besides, we can easily prove that **RRSplit** finds the maximum common subgraph based on our discussion above. Finally, we analyze the space complexity and time complexity of **RRSplit** as below.

Algorithm 2: Our proposed algorithm: **RRSplit**

Input: Two graphs $Q = (V_Q, E_Q)$ and $G = (V_G, E_G)$

Output: The maximum common subgraph

```

1  $S^* \leftarrow \emptyset$ ; // Global variable
2 RRSplit-Rec( $\emptyset, V_Q \times V_G, \emptyset$ );
3 Return  $S^*$ ;
4 Procedure RRSplit-Rec( $S, C, D$ )
5   if  $|S| > |S^*|$  then  $S^* \leftarrow S$ ;
6   /* Termination (Lemma 5) */
7   if  $C = \emptyset$  then return;
8   if  $ub_{S,C,D} \leq |S^*|$  then return;
9   /* Branching */
10  Select a branching vertex  $u$  and a branching subset  $X \times Y$  from  $\mathcal{P}(C)$  based on a policy;
11  /* Maximality-based reduction */
12  if there exists a vertex  $v$  in  $Y$  such that  $\langle u, v \rangle$  satisfies Equation (13) then
13    RRSplit-Rec( $S \cup \{\langle u, v \rangle\}, C \setminus u \setminus v, D \cup \{u\} \times (Y \setminus \{v\})$ );
14    return;
15  /* Branching at the first group */
16   $Y_{temp} \leftarrow Y$ ;
17  for  $i = 1, 2, \dots, |Y|$  do
18    Select and remove a vertex  $v$  from  $Y_{temp}$  based on a policy;
19    if there exists a vertex pair  $\langle u', v \rangle$  in  $D$  such that  $u' \in \Psi(u)$  then
20      continue;
21    Refine candidate set  $C \setminus u \setminus v$  as  $C_i$  based on Equation (4);
22    RRSplit-Rec( $S \cup \{\langle u, v \rangle\}, C_i, D \cup \{u\} \times (Y \setminus Y_{temp})$ );
23  /* Branching at the second group */
24  RRSplit-Rec( $S, C \setminus \Psi(u), D$ );

```

Space complexity. We note that **RRSplit** recursively maintains three global data structures, namely S , C and D , for each branch, which dominate the space complexity of **RRSplit**. Let S^* be the largest common subgraph between graphs Q and G . First, partial solution S is a set of vertex pairs and its size is bounded by $O(|S^*|)$. Second, candidate set C is also a set of vertex pairs and can be partitioned as several subsets, i.e., $C = X_1 \times Y_1 \cup X_2 \times Y_2 \cup \dots \cup X_c \times Y_c$ where c is a positive integer, based on Equation (4). We note that subsets in $X_1, X_2, \dots, X_c$ (resp. $Y_1, Y_2, \dots, Y_c$) are mutually disjoint and $X_1 \cup X_2 \cup \dots \cup X_c = X$ (resp. $Y_1 \cup Y_2 \cup \dots \cup Y_c = Y$), as discussed in the proof of Lemma 2.

Therefore, C can be stored as c subsets, each of which $\langle X_i, Y_i \rangle$ ($1 \leq i \leq c$) consists of two sets X_i and Y_i . Thus, the size of C is bounded by $O(|V_Q| + |V_G|)$. Third, D is a set of vertex pairs and consists of at most $|S^*| \cdot |V_G|$ different vertex pairs since for a vertex pair $\langle u, v \rangle$ in D , (1) u must appear in S based on our maintenance of D and thus has at most $|S^*|$ different values and (2) v has at most $|V_G|$ different values clearly. In summary, the space complexity of `RRSplit` is $O(|V_Q| + |S^*| \times |V_G|)$.

Time complexity of the proposed reductions. First, the reduction at the first group takes $O(|V_Q| + |V_G|)$ time (Lines 15-16). In specific, D is organized as several disjoint subsets, i.e., $D = \{u_1\} \times A_1 \cup \{u_2\} \times A_2 \cup \dots \cup \{u_d\} \times A_d$ where d is a positive integer. Thus, it can be conducted in two steps: (1) for each vertex u_i appearing in D , it takes $O(1)$ to check whether $u_i \in \Psi(u)$ and (2) if $u_i \in \Psi(u)$, it takes $O(|A_i|)$ to check whether $\langle u_i, v \rangle \in \{u_i\} \times A_i$. We note that for any two distinct vertices u_i and u_j appearing in D such that $u_i \in \Psi(u_j)$, it is no hard to verify that $A_i \cap A_j = \emptyset$ due to the reduction at the first group (for which we put the details of the proof in the technical report [53]). As a result, we have $\sum_{u_i \in \Psi(u)} |A_i| \leq |V_G|$. Second, the reduction at the second group runs in $O(|X|)$ for updating $C \setminus \Psi(u)$ at Line 19, which is bounded by $O(|V_Q|)$. In specific, it can be done by removing from X all vertices in $\Psi(u)$ (note that, given all structurally equivalent classes, determining whether a vertex belongs to $\Psi(u)$ can be done in $O(1)$). Third, the maximality reduction runs in $O(\sum_{\langle X', Y' \rangle \in C} |X'| + |Y'| \cdot |Y|)$, which is bounded by $O(|V_Q| + |V_G|^2)$. In specific, for each vertex in $|Y|$, it needs to check the condition in Equation (13). Fourth, the new upper bound can be obtained in $O(|V_Q| + |V_G|^2)$. In specific, the time cost is dominated by the computation of $ub_{X', Y', D}$ for each subset $\langle X', Y' \rangle$ in C . $ub_{X', Y', D}$ can be obtained in $O(|X'| + \sum_{u_i \in \Psi(u')} |A_i| + |Y'|)$, where u' is a random vertex selected from X' and $\{u_i\} \times A_i$ is a subset in D , which is bounded by $O(|X'| + |V_G|)$. Therefore, the new upper bound can be obtained in $O(\sum_{X' \times Y' \in C} (|X'| + |V_G|))$, which is bounded by $O(|V_Q| + |V_G|^2)$.

Worst-case time complexity of `RRSplit`. We note that the worst-case time complexity of `RRSplit` is dominated by the number of recursive calls of `RRSplit-Rec` (i.e., the number of formed branches) since `RRSplit-Rec` runs in polynomials of $|V_Q|$ and $|V_G|$. Formally, we have the following theorem.

THEOREM 6. *Assume that $|V_Q| \leq |V_G|$. The worst-case time complexity of our proposed `RRSplit` is $O^*((|V_G| + 1)^{|V_Q|})$, where $O^*(\cdot)$ suppresses the polynomials.*

PROOF. It is easy to verify that the worst-case time complexity of `RRSplit` is bounded by the number of branches. Consider a branch $B = (S, C, D)$. For all sub-branches formed at B by selecting a branching vertex u^* , we observe that only the sub-branch in the second group has the same partial solution S with B . Based on this, we can easily deduce that there are at most $|V_Q|$ branches which share the same partial solution. Besides, we observe that each vertex in $V_p \cup V_q$ only appears in one pair of S , i.e., for any two distinct pairs $\langle u, v \rangle$ and $\langle u', v' \rangle$ in S , we have $u \neq u'$ and $v \neq v'$. Based on this, let $|S| = k$ where $0 \leq k \leq |V_Q|$, and we can deduce that there are at most $k! \binom{|V_Q|}{k} \binom{|V_G|}{k}$ different partial solutions with the size of k by applying the multiplication principle (note that V_p has $\binom{|V_Q|}{k}$ different choices, V_q has $\binom{|V_G|}{k}$ different choices, and the bijection ϕ between V_p and V_q has $k!$ different choices). Therefore, the number of branches is at most

$$T = |V_Q| \sum_{k=0}^{|V_Q|} k! \binom{|V_Q|}{k} \binom{|V_G|}{k}. \quad (20)$$

Table 1. Datasets used in the experiments (“# of solved instances” refers to the number of instances solved by algorithms within 1,800 seconds and “Achieved speedups” refers to the percentage of the solved instances that RRSplit runs at least 5×/10×/100× faster than McSplitDAL)

Dataset	Domain	# of graphs	# of instances	# of vertices	# of solved instances		Achieved speedups		
					RRSplit	McSplitDAL	5×	10×	100×
BI	Biochemical	136	9,180	9~ 386	7,730	4,696	91.3%	84.4%	69.7%
CV	Segmented images	190	6,424	22~ 5,972	1,351	1,291	76.5%	48.6%	0.2%
PR	Segmented images	25	24	4~ 4,838	24	24	91.7%	91.7%	58.3%
LV	Synthetic	112	6,216	10~ 6,671	1,059	883	68.0%	54.7%	38.3%

We then show that T is bounded by $O^*((|V_G| + 1)^{|V_Q|})$ as below.

$$T = |V_Q| \sum_{k=0}^{|V_Q|} (|V_Q| - k)! \binom{|V_Q|}{k} \binom{|V_G|}{|V_Q| - k} \quad (21)$$

$$= |V_Q| \sum_{k=0}^{|V_Q|} \frac{(|V_G|)!}{(|V_G| - |V_Q| + k)!} \binom{|V_Q|}{k} \quad (22)$$

$$\leq |V_Q| \sum_{k=0}^{|V_Q|} (|V_G|)^{|V_Q| - k} \binom{|V_Q|}{k} = |V_Q| (|V_G| + 1)^{|V_Q|}, \quad (23)$$

where $(|V_G|)! / (|V_G| - |V_Q| + k)!$ is much smaller than $(|V_G|)^{|V_Q| - k}$ clearly and $(|V_G| + 1)^{|V_Q|}$ in the last equation is derived by the binomial theorem. $\square$

Remark. Note that the assumption that $|V_Q| \leq |V_G|$ is not a restrictive assumption: it is realistic in practice. We remark that to our best knowledge, the achieved worst-case time complexity $O^*((|V_G| + 1)^{|V_Q|})$ of RRSplit matches the best-known worst-case time complexity for the problem [47]. However, the algorithm proposed in [47] is of theoretical interest only and is not practically efficient. Besides, we note that McSplit and its variants [30, 31, 33, 55] do not have any theoretical guarantees on the worst-case time complexity.

5 Experiments

Datasets. Following existing studies [20, 30, 31, 33, 42, 55], we use four benchmark graph collections, namely biochemicalReactions (BI), images-CVIU11 (CV), images-PR15 (PR) and LV (LV), in the experiments. All datasets are collected from <http://liris.cnrs.fr/csolnon/SIP.html> and come from real-world applications in various domains, as shown in Table 1. Specifically, BI contains 136 unlabeled bipartite graphs, each of which corresponds to a biochemical reaction network. CV contains 44 pattern graphs and 146 target graphs, which are generated from segmented images. PR contains 24 pattern graphs and 1 target graph, which are also from segmented images. LV contains 112 graphs generated from biological networks. All graphs have up to thousands of vertices. We note that (1) solving our problem on two graphs with beyond 10K vertices is challenging based on the worst-case time complexity of $O^*((|V_G| + 1)^{|V_Q|})$, (2) the largest graph used in previous studies [30, 31, 33, 55] has 6,771 vertices, which is also covered (in LV) by our experiments, and (3) finding the largest common subgraph between two graphs with thousands of vertices has found many real applications [15]. Following existing studies [20, 30, 31, 33, 42, 55], for BI and LV, we generate and test the problem instances (i.e., Q and G) by pairing any two distinct graphs; and for CV and PR which consist of two types of graphs, namely pattern graphs and target graphs, we test all those problem instances with one graph Q from pattern graphs and the other G from target graphs.

Table 2. Comparison of running time on all datasets (statistics of achieved speedups in Figure 5)

Dataset	RRSplit runs faster			McSplitDAL runs faster		
	% of instances	Avg. speedup	Max. speedup	% of instances	Avg. speedup	Max. speedup
BI	99.43%	3.3×10^4	10^6	0.5%	24.81	872.37
CV	92.15%	10.92	161	7.84%	4.96	38.97
PR	95.83%	139.39	234	4.17%	1.23	1.23
LV	93.48%	1.2×10^4	10^6	6.51%	24.23	652.13

Algorithms. We compare the newly proposed algorithm RRSplit with McSplitDAL [31]. To be specific, McSplitDAL is one variant of McSplit as introduced in Section 3, which follows the framework of McSplit (i.e., Algorithm 1) and introduces some learning-based techniques for optimizing the policies of selecting vertices at line 6, line 8 and line 10 of Algorithm 1. To our best knowledge, McSplitDAL is the state-of-the-art algorithm and runs significantly faster than previous solutions, including McSplitLL [55] and McSplitRL [31]. Besides these, in order to study the effectiveness of different reductions employed in our algorithm RRSplit, we evaluate three variants of RRSplit – RRSplit-VE, RRSplit-MB, and RRSplit-UB, respectively obtained by turning off vertex-equivalence based reductions, maximality based reductions, and vertex-equivalence based upper bound.

Implementation and metrics. All algorithms are implemented in C++ and compiled with -O3 optimization. All experiments run on a Linux machine with a 2.10GHz Intel CPU and 128GB memory. Note that, for the implementation of McSplitDAL, we directly use the source code from the authors of [31]. We record and compare the total running times of the algorithms on different problem instances (note that the measured running time excludes the I/O time of reading graphs from the disk). We set the running time limit (INF) as 1,800 seconds by default. Our data and code are available at <https://github.com/KaiqiangYu/SIGMOD25-MCSS>.

5.1 Comparison among algorithms

All datasets (running time). We compare our algorithm RRSplit with the baseline McSplitDAL on all graph collections. Following some existing works [32], we report the running times of the algorithms on various problem instances in Figure 5. Specifically, each dot in the scatter figures represents a problem instance, with the x -axis (resp. y -axis) corresponding to the running time of RRSplit (resp. McSplitDAL) on the instance. Hence, for those problem instances with small values on x -axis and large values on y -axis (which thus locate on the top left region of the figures), RRSplit performs better than McSplitDAL. We mark the running time as INF if the problem instance cannot be solved within the default time limit. Besides, we also provide some statistics in Table 1 and Table 2. We observe that (1) RRSplit outperforms McSplitDAL by achieving around one to four orders of magnitude speedup (in average) on the majority (above 92%) of the tested problem instances and (2) McSplitDAL cannot handle all problem instances within the time limit. We do note that McSplitDAL runs slightly faster on a few (below 8%) problem instances in CV and LV. Some possible reasons are as follows. First, our RRSplit introduces some extra time costs for conducting the proposed reductions as well as computing the upper bound. Second, the heuristic policies adopted in RRSplit and McSplitDAL for branching may have different behaviors. In specific, on these problem instances, the heuristic policies may help McSplitDAL to find a large common subgraph quickly so as to prune more unpromising branches via the upper-bound based reduction (note that they are based on reinforcement learning and the behaviors of the learned policy is based on the explored branches during the running time).

All datasets (number of formed branches). We report the number of branches formed by the algorithms on different problem instances in Figure 6. Similarly, each dot in the scatter figures

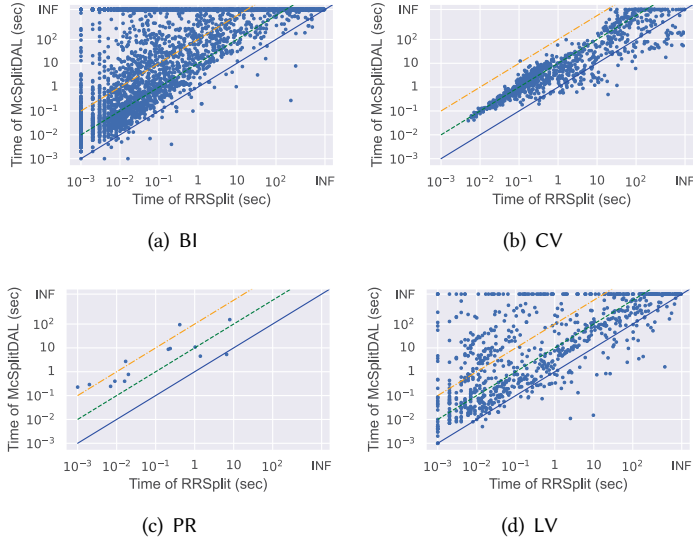


Fig. 5. Running time on all datasets. For those problem instances locating at the right side of dash line ‘-’ with orange color (resp. ‘-’ with green color), RRSplit achieves at least 100 $\times$ (resp. 10 $\times$) speedup compared with McSplitDAL.

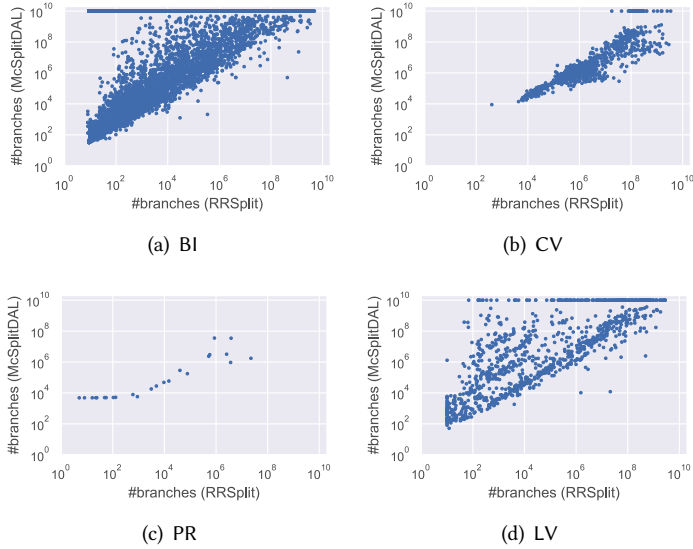


Fig. 6. Number of formed branches on all datasets

represents a problem instance, with the x -axis (resp. y -axis) corresponding to the number of branches formed by RRSplit (resp. McSplitDAL) on the instance. We have the following observations. First, the number of branches formed by RRSplit is significantly smaller than that formed by McSplitDAL, e.g., the former is around 10% - 0.01% of the latter on the most of problem instances. This shows the effectiveness of our proposed maximality-based reductions and vertex-equivalence-based

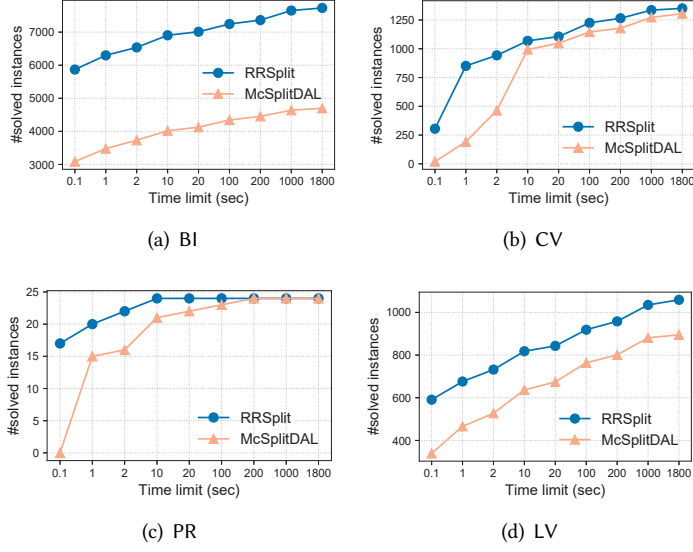


Fig. 7. Comparison by varying time limits

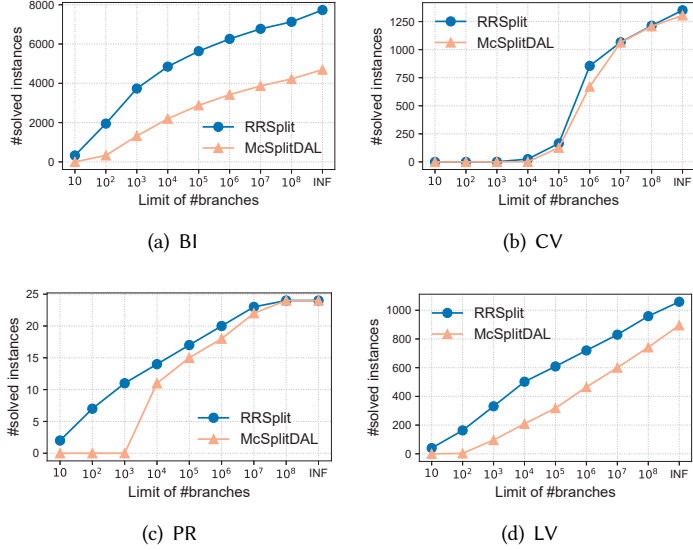


Fig. 8. Comparison by varying the limit of number of formed branches

reductions. Second, the distribution of the number of formed branches in Figure 6 is consistent with that of the running time in Figure 5. This indicates the achieved speedups on the running time can be traced to our newly-designed reductions.

Varying time limits. We report the number of solved problem instances in Figure 7 as the time limit is varied. Clearly, all algorithms solve more problem instances as the time limit increases. We observe that RRSPplit solves more problem instances than McSPplitDAL within the same time

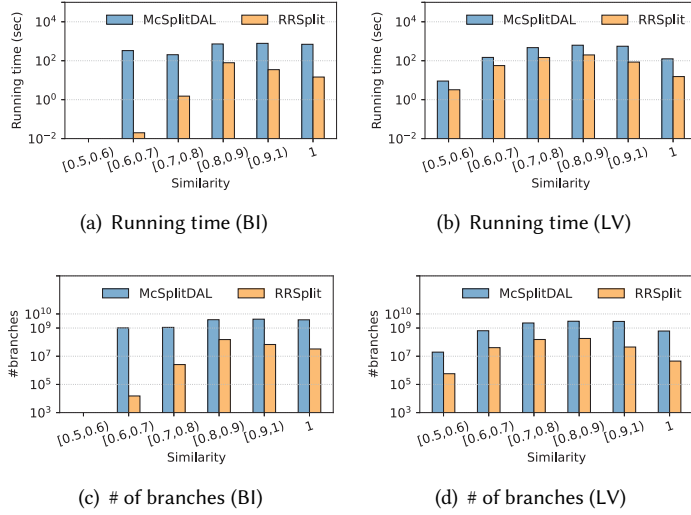


Fig. 9. Comparison by varying similarities

limit. In particular, RRSplit with a time limit of 1 second even solves more problem instances than McSplitDAL with a time limit of 10 seconds in all graph collections except for CV; and on PR, RRSplit solves all problem instances within the time limit of 10 seconds. This further demonstrates the superiority of our algorithm RRSplit over the baseline McSplitDAL.

Varying the limits of number of formed branches. We report the number of solved problem instances in Figure 8 as the limit on number of formed branches is varied. We note that the more branches are allowed to be formed, the more instances will be solved. We observe that (1) RRSplit solves more problem instances than McSplitDAL within the same limit of the number of formed branches and (2) the results in Figure 8 show similar tendencies as those in Figure 7 in general. This further explains the practical superiority of the newly proposed reductions.

Varying the similarities of input graphs. We define the similarity of input graphs Q and G , $Sim(Q, G)$, as follows.

$$Sim(Q, G) = \frac{|S^*|}{\min\{|V_Q|, |V_G|\}}, \quad (24)$$

where S^* is the maximum common subgraph between Q and G . Clearly, $Sim(Q, G)$ varies from 0 to 1, and the larger the value of $Sim(Q, G)$, the higher the similarity between Q and G . We test different problem instances as the similarity varies from 0.5 to 1 on BI and LV, and report the average running time in Figures 9(a)-(b) and the average number of formed branches in Figures 9(c)-(d). The results on CV and PR show similar trends, complete details of which appear in the technical report [53]. We can see that RRSplit consistently outperforms McSplitDAL in various settings, e.g., RRSplit runs several orders of magnitude faster and forms fewer branches than McSplitDAL. This demonstrates that our designed reductions are effective for pruning the redundant branches on problem instances with various similarities. Besides, we observe that both RRSplit and McSplitDAL have the running time and the number of formed branches first increase and then decrease as the similarity grows. The possible reasons are as follows. (1) The maximum common subgraphs become larger as the similarity increases according to Equation (24) and typically more common subgraphs will be explored for finding a large maximum common subgraph. Therefore, the running time

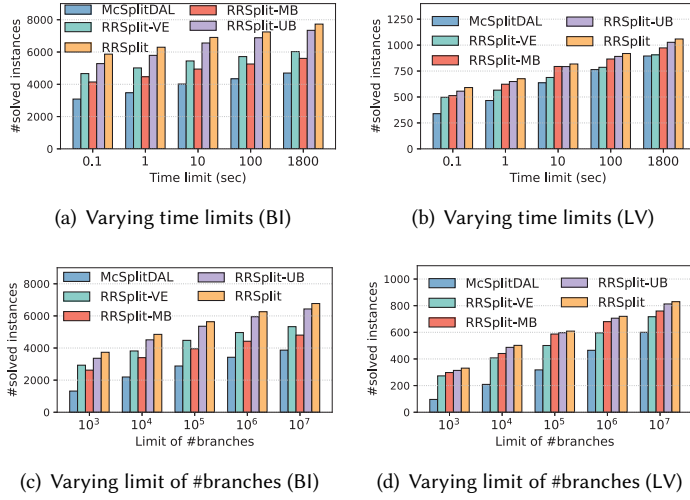


Fig. 10. Comparison among various reductions

firstly increases; (2) the upper-bound based reduction performs better as the similarity grows. For example, in the setting of $\text{Sim}(Q, G) = 1$, the algorithm can be terminated directly once a common subgraph with $\min\{|V_Q|, |V_G|\}$ vertices is found. Therefore, the running time then decreases.

Scalability test. We test the scalability of our RRSplit on two large datasets, i.e., Twitter and DBLP, which are collected from different domains (<http://konect.cc/>). Here, Twitter is a social network with 465,017 vertices and 833,540 edges, and DBLP is a collaboration network with 317,080 vertices and 1,049,866 edges. Following existing studies [3, 21, 45], we generate the problem instances (i.e., Q and G) as follows. Let Twitter or DBLP be the graph G . We first extract a set of graphs Q from G . Specifically, we conduct a random walk on G and extract a subgraph induced by the visited vertices. By varying the size of the extracted graph Q (among $\{20, 30, 40, 50, 60\}$), we extract 5 sets and each of them contains 100 different graphs Q . Then, we generate different problems by pairing the graph G (i.e., Twitter or DBLP) with different graphs Q in the set. In summary, for each dataset, we have 500 different problem instances.

We compare our RRSplit with McSplitDAL by varying the size of Q , and report the average running time in Figure 11. We observe that our RRSplit outperforms McSplitDAL significantly. Besides, McSplitDAL cannot handle almost all the problem instances within the time and/or space limit (INF/OOM). This is because the implementation of McSplitDAL highly relies on the adjacent matrix of Q and G , which introduces huge space and time costs when G is very large. Finally, we observe that RRSplit has the running time increase as the size of Q grows. This is also consistent with the theoretical analysis.

5.2 Ablation studies

We study the effects of various reductions on reducing the redundant computations. In specific, we compare RRSplit with three variants, namely RRSplit-VE: the full version without vertex-equivalence based reductions, RRSplit-MB: the full version without maximality based reductions and RRSplit-UB: the full version without the vertex-equivalence based upper bound, on BI and LV. We report the number of solved problem instances in Figure 10(a,b) for varying the time limit and in Figure 10(c,d) for varying the limit of number of formed branches. The results on CV

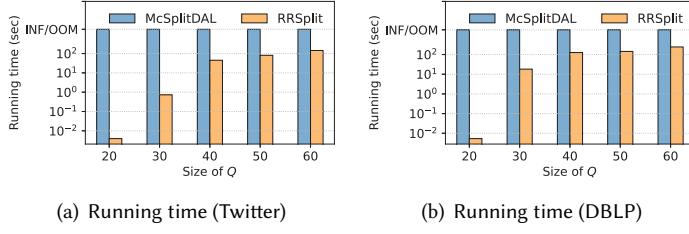


Fig. 11. Scalability test on large datasets

and PR show similar clues, which we put in the technical report [53]. First, we can see that all four algorithms perform better than the baseline McSplitDAL, among which RRSplit performs the best. This demonstrates the effectiveness of vertex-equivalence-based reductions, maximality-based reductions and vertex-equivalence-based upper bound. Second, RRSplit-VE and RRSplit-MB achieve comparable performance and both contribute to the improvements. Specifically, we note that RRSplit-VE runs slightly faster than RRSplit-MB on BI while RRSplit-MB runs slightly faster than RRSplit-VE on LV. This is possibly because graphs in BI are relatively small biochemical networks where two vertices are more likely to be structural equivalent and thus the vertex-equivalence based reductions outperform other reductions, while graphs in LV are synthetic networks.

6 Related Work

Maximum common subgraph search. In the literature, there are quite a few studies on finding the maximum common subgraph, which solve the problem either exactly [1, 26, 28, 30–34, 47, 49, 55] or approximately [4, 13, 39, 50, 54]. First, among all those exact algorithms, they mainly focus on improving the *practical* performance and most of them are backtracking (also known as branch-and-bound) algorithms [28, 34]. Specifically, authors in [28, 34] propose the first backtracking framework. The idea is to transform the problem of finding the maximum common subgraph between two given graphs to the problem of finding the maximum clique in the *association graph*. Then, authors in [32, 49] follow the previous framework and further improve it by employing the constraint programming techniques. However, these algorithms are all based on a large and dense association graph built from two given graphs, which thus suffer from the efficiency issue. To solve the issue, McCreesh et al. [33] propose a new backtracking framework, namely McSplit, which is not based on the maximum clique search problem. Recent works [30, 31, 55] follow McSplit and improve the practical performance by optimizing the policies of branching via learning techniques. Among them, McSplitDAL [31] runs faster than others. We note that some exact algorithms are designed to achieve improvements of theoretical time complexity [1, 26, 28, 47]. They have gradually improved the worst-case time complexity from $O^*(1.19^{|V_Q|+|V_G|})$ [28] to $O^*(|V_Q|^{(|V_G|+1)})$ [26], and to $O^*((|V_Q|+1)^{|V_G|})$ [47], which is our best-known worst-case time complexity for the problem. However, these algorithms are of theoretical interests only and not efficient in practice. We remark that (1) our RRSplit not only runs faster than all previous algorithms in practice but also achieves the state-of-the-art worst case time complexity (i.e., $O^*((|V_Q|+1)^{|V_G|})$) in theory and (2) the heuristic policies proposed in [30, 31, 55] are orthogonal to RRSplit. Second, since the problem of finding the largest common subgraph is NP-hard, some researchers turn to solve it approximately in polynomial time. Some approximation algorithms include meta-heuristics [13, 39], spectra methods [50], and learning-based methods [4, 54]. We remark that these techniques cannot be applied to our exact algorithm directly.

Subgraph matching. Given a target graph and a query graph, subgraph matching aims to find from a target graph all those subgraphs isomorphic to a query graph. We note that maximum common subgraph search is a generalization of subgraph matching. Specifically, given two graphs Q and G , maximum common subgraph search would reduce to subgraph matching if we require that the found common subgraph has the size at least $|V(Q)|$ or $|V(G)|$. In recent decades, subgraph matching has been widely studied [3, 6, 7, 14, 17, 18, 21, 24, 41, 43–45, 48]. The majority of proposed solutions perform a backtracking search. Among these algorithms, the *candidate filtering* technique, which is designed for removing unnecessary vertices from the target graph, has been shown to be important for improving the practical efficiency [6, 7, 17, 18, 24]. The technique relies on an auxiliary data structure (e.g., a tree or a directed acyclic graph), which is obtained from the query graph (based on the implicit constraint that each vertex in the query graph must be mapped to a vertex in the found subgraph). We note that it is hard to apply candidate filtering to find the maximum common subgraph (since the mentioned constraint may not hold). We remark that finding subgraphs exactly isomorphic to a query graph is too restrictive in some real applications due to the data quality issues and/or potential requirements of the fuzzy search (e.g., no result would be returned if there does not exist any subgraph isomorphic to a query graph). Motivated by this, we focus on finding the maximum common subgraph between two graphs in this paper.

7 Conclusion

In this paper, we propose a new backtracking algorithm `RRSplit` for finding the largest common subgraph. `RRSplit` is based on our newly-designed reduction rules for reducing the redundant computations and achieves the state-of-the-art worst-case time complexity. Extensive experiments are conducted on the widely-used graph collections, and the results demonstrate the superiority of our method. In the future, we will adapt our proposed algorithm to solve the maximum common subgraph search problem on other types of graphs, including vertex-labeled and edge-labeled graphs.

8 Acknowledgement

The research of Kaiqiang Yu, Kaixin Wang and Cheng Long is supported by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Award MOE-T2EP20221-0013 and Tier 1 Awards RG20/24 and RG77/21)). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore. Lakshmanan's research was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada (Grant Number RGPIN-2020-05408). Reynold Cheng was supported by the Hong Kong Jockey Club Charities Trust (Project 260920140), the University of Hong Kong (Project 2409100399), the HKU Outstanding Research Student Supervisor Award 2022-23, and the HKU Faculty Exchange Award 2024 (Faculty of Engineering).

References

- [1] Faisal N Abu-Khzam. 2014. Maximum common induced subgraph parameterized by vertex cover. *Inform. Process. Lett.* 114, 3 (2014), 99–103.
- [2] Aurelio Antelo-Collado, Ramón Carrasco-Velaz, Nicolás García-Pedrajas, and Gonzalo Cerruela-García. 2020. Maximum common property: a new approach for molecular similarity. *Journal of cheminformatics* 12 (2020), 1–22.
- [3] Junya Arai, Yasuhiro Fujiwara, and Makoto Onizuka. 2023. GuP: Fast Subgraph Matching by Guard-based Pruning. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.
- [4] Yunsheng Bai, Derek Xu, Yizhou Sun, and Wei Wang. 2021. Gsearch: Maximum common subgraph detection via learning to search. In *International Conference on Machine Learning*. PMLR, 588–598.
- [5] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V Hicks. 2011. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research* 59, 1 (2011), 133–142.
- [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*. 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*. 1199–1214.
- [8] 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 (2013), 1–13.
- [9] H. Bunke. 1997. On a relation between graph edit distance and maximum common subgraph. *Pattern recognition letters* 18, 8 (1997), 689–694.
- [10] Lijun Chang, Xing Feng, Xuemin Lin, Lu Qin, Wenjie Zhang, and Dian Ouyang. 2020. Speeding up GED verification for graph similarity search. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 793–804.
- [11] Xiaoyang Chen, Hongwei Huo, Jun Huan, and Jeffrey Scott Vitter. 2019. An efficient algorithm for graph edit distance computation. *Knowledge-Based Systems* 163 (2019), 762–775.
- [12] Tony Chiang, Denise Scholtens, Deepayan Sarkar, Robert Gentleman, and Wolfgang Huber. 2007. Coverage and error models of protein-protein interaction data by directed graph analysis. *Genome biology* 8 (2007), 1–14.
- [13] Jaewon Choi, Yourim Yoon, and Byung-Ro Moon. 2012. An efficient genetic algorithm for subgraph isomorphism. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 361–368.
- [14] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence* 26, 10 (2004), 1367–1372.
- [15] Hans-Christian Ehrlich and Matthias Rarey. 2011. Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 1, 1 (2011), 68–79.
- [16] Karam Gouda and Mosab Hassaan. 2016. CSI_GED: An efficient approach for graph edit similarity computation. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 265–276.
- [17] 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 *Proceedings of the 2019 International Conference on Management of Data*. 1429–1446.
- [18] 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*. 337–348.
- [19] Avik Hati, Subhasis Chaudhuri, and Rajbabu Velmurugan. 2016. Image co-segmentation using maximum common subgraph matching and region co-growing. In *Computer Vision—ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VI* 14. Springer, 736–752.
- [20] Ruth Hoffmann, Ciaran McCreesh, and Craig Reilly. 2017. Between subgraph isomorphism and maximum common subgraph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.
- [21] Tatiana Jin, Boyang Li, Yichao Li, Qihui Zhou, Qianli Ma, Yunjian Zhao, Hongzhi Chen, and James Cheng. 2023. Circinus: Fast redundancy-reduced subgraph matching. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.
- [22] Viggo Kann. 1992. On the approximability of the maximum common subgraph problem. In *STACS 92: 9th Annual Symposium on Theoretical Aspects of Computer Science Cachan, France, February 13–15, 1992 Proceedings* 9. Springer, 375–388.
- [23] 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*. 925–937.
- [24] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong, and Wook-Shin Han. 2023. Fast subgraph query processing and subgraph matching via static and dynamic equivalences. *The VLDB journal* 32, 2 (2023), 343–368.

- [25] Jongik Kim. 2023. Efficient graph edit distance computation using isomorphic vertices. *Pattern Recognition Letters* 168 (2023), 71–78.
- [26] Evgeny B Krissinel and Kim Henrick. 2004. Common subgraph isomorphism detection by backtracking search. *Software: Practice and Experience* 34, 6 (2004), 591–607.
- [27] Simon J Larsen and Jan Baumbach. 2017. CytoMCS: a multiple maximum common subgraph detection tool for Cytoscape. *Journal of integrative bioinformatics* 14, 2 (2017), 20170014.
- [28] Giorgio Levi. 1973. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo* 9, 4 (1973), 341–352.
- [29] Harry R Lewis. 1983. Michael R. Garey and David S. Johnson. Computers and intractability. A guide to the theory of NP-completeness. WH Freeman and Company, San Francisco 1979, x+ 338 pp. *The Journal of Symbolic Logic* 48, 2 (1983), 498–500.
- [30] Yanli Liu, Chu-Min Li, Hua Jiang, and Kun He. 2020. A learning based branch and bound for maximum common subgraph related problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 2392–2399.
- [31] Yanli Liu, Jiming Zhao, Chu-Min Li, Hua Jiang, and Kun He. 2023. Hybrid learning with new value function for the maximum common induced subgraph problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 4044–4051.
- [32] Ciaran McCreesh, Samba Ndojh Ndiaye, Patrick Prosser, and Christine Solnon. 2016. Clique and constraint models for maximum common (connected) subgraph problems. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 350–368.
- [33] Ciaran McCreesh, Patrick Prosser, and James Trimble. 2017. A partitioning algorithm for maximum common subgraph problems. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. 712–719.
- [34] James J McGregor. 1982. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience* 12, 1 (1982), 23–34.
- [35] Thien Nguyen, Dominic Yang, Yurun Ge, Hao Li, and Andrea L Bertozzi. 2019. Applications of structural equivalence to subgraph isomorphism on multichannel multigraphs. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 4913–4920.
- [36] Parisutham Nirmala, Ramasubramony Sulochana Lekshmi, and Rethnasamy Nadarajan. 2016. Vertex cover-based binary tree algorithm to detect all maximum common induced subgraphs in large communication networks. *Knowledge and Information Systems* 48 (2016), 229–252.
- [37] Younghee Park and Douglas Reeves. 2011. Deriving common malware behavior through graph clustering. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. 497–502.
- [38] Chengzhi Piao, Tingyang Xu, Xiangguo Sun, Yu Rong, Kangfei Zhao, and Hong Cheng. 2023. Computing graph edit distance via neural graph matching. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1817–1829.
- [39] Jochem H Rutgers, Pascal T Wolkotte, Philip KF Hölzenspies, Jan Kuper, and Gerard JM Smit. 2010. An approximate maximum common subgraph algorithm for large digital circuits. In *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. IEEE, 699–705.
- [40] Robert Schmidt, Florian Krull, Anna Lina Heinzke, and Matthias Rarey. 2020. Disconnected maximum common substructures under constraints. *Journal of Chemical Information and Modeling* 61, 1 (2020), 167–178.
- [41] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. 2008. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment* 1, 1 (2008), 364–375.
- [42] Christine Solnon, Guillaume Damiand, Colin De La Higuera, and Jean-Christophe Janodet. 2015. On the complexity of submap isomorphism and maximum common submap problems. *Pattern Recognition* 48, 2 (2015), 302–316.
- [43] Shixuan Sun and Qiong Luo. 2020. Subgraph matching with effective matching order and indexing. *IEEE Transactions on Knowledge and Data Engineering* 34, 1 (2020), 491–505.
- [44] Shixuan Sun, Xibo Sun, Yulin Che, Qiong Luo, and Bingsheng He. 2020. Rapidmatch: A holistic approach to subgraph query processing. *Proceedings of the VLDB Endowment* 14, 2 (2020), 176–188.
- [45] Xibo Sun and Qiong Luo. 2023. Efficient GPU-Accelerated Subgraph Matching. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.
- [46] Yi Sun, Ali Kashif Bashir, Usman Tariq, and Fei Xiao. 2021. Effective malware detection scheme based on classified behavior graph in IIoT. *Ad Hoc Networks* 120 (2021), 102558.
- [47] W Henry Suters, Faisal N Abu-Khzam, Yun Zhang, Christopher T Symons, Nagiza F Samatova, and Michael A Langston. 2005. A new approach and faster exact methods for the maximum common subgraph problem. In *Computing and Combinatorics: COCOON 2005*. Springer, 717–727.
- [48] Julian R Ullmann. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)* 23, 1 (1976), 31–42.
- [49] Philippe Vismara and Benoît Valery. 2008. Finding maximum common connected subgraphs using clique detection or constraint satisfaction algorithms. In *International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences*. Springer, 358–368.

- [50] Bai Xiao, Edwin R Hancock, and Richard C Wilson. 2009. A generative model for graph matching and embedding. *Computer Vision and Image Understanding* 113, 7 (2009), 777–789.
- [51] Xifeng Yan, Philip S Yu, and Jiawei Han. 2005. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 766–777.
- [52] Dominic Yang, Yurun Ge, Thien Nguyen, Denali Molitor, Jacob D Moorman, and Andrea L Bertozzi. 2023. Structural Equivalence in Subgraph Matching. *IEEE Transactions on Network Science and Engineering* (2023).
- [53] Kaiqiang Yu, Kaixin Wang, Cheng Long, Laks Lakshmanan, and Reynold Cheng. 2025. Fast Maximum Common Subgraph Search: A Redundancy-Reduced Backtracking Approach (Technical report). <https://arxiv.org/pdf/2502.11557>.
- [54] Andrei Zanfir and Cristian Sminchisescu. 2018. Deep learning of graph matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2684–2693.
- [55] Jianrong Zhou, Kun He, Jiongzhi Zheng, Chu-Min Li, and Yanli Liu. 2022. A Strengthened Branch and Bound Algorithm for the Maximum Common (Connected) Subgraph Problem. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022*. 1908–1914.

Received October 2024; revised January 2025; accepted February 2025