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

ABSTRACT

Finding the largest subgraph that commonly occurs in two given graphs, namely 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 interests. Specifically, those algorithms with theoretical guarantees on the running time are not practically efficient; others, 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 both achieves better practical efficiency and provides the non-trivial theoretical guarantees on the worst-case time complexity. In specific, to achieve the former, we develop a series of reductions and upper bound for reducing redundant computations, i.e., the time costs for exploring some unpromising branches that hold no maximum common subgraph. To achieve the latter, we formally prove that RRSplit achieves the worst-case time complexity which matches the best-known worst-case time complexity for the problem. Finally, we conduct extensive experiments on four benchmark graph collections, and the results demonstrate that our algorithm outperforms the practical state-of-the-art by several orders of magnitude.

ACM Reference Format:
. 2018. Fast Maximum Common Subgraph Search: A Redundancy-Reduced Backtracking Approach. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 16 pages. https://doi.org/XXXXXXX.XXXXXXX

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 and 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 (induced) subgraph search*, which is widely used to measure the similarity of two graphs [4, 10, 25–29, 33, 43, 44, 47, 48]. More specifically, a common (induced) 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 an induced subgraph q of Q and an induced subgraph g of G, and a bijection mapping $\phi : V_q \rightarrow V_g$ such that q and g are *isomorphic* to each other under the bijection ϕ, which we denote by $\langle q, g, \phi \rangle$. Given two graphs Q and G, the problem aims to find the common subgraph $\langle q, g, \phi \rangle$ with the maximum number of vertices in q and g.

The maximum common subgraph search problem has many applications across various disciplines, and has been widely studied [1, 4, 10, 21, 23, 25–29, 33, 41, 43, 44, 47, 48]. To be specific, it offers an operator for evaluating the similarity among graphs in graph database system [45] and thus has found a wide range of real applications, including cheminformatics [2, 34], communication networks [31], software analysis [32, 40], biochemistry [8, 12, 22], and image segmentation [15]. For example, the similarity $Sim(Q, G)$ between two molecules Q and G is calculated based on the maximum common subgraph $\langle q, g, \phi \rangle$ between Q and G, i.e., $Sim(Q, G) = (|V_q| + |E_q|)/((|V_Q| + |V_G|) \times (|E_Q| + |E_G|))$ [12]. 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 [12]. Besides, the maximum common subgraph search problem generalizes the well-studied *subgraph matching* problem [3, 6, 7, 11, 13, 14, 17, 20, 35, 37–39, 42]. More specifically, given a data graph G and a query graph Q, subgraph matching seeks to find if there is an embedding[1] 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, 9] 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. 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 [24] 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 $r \geq 1$ [18]. 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, 21, 23, 41]. They have gradually improved the worst-case time complexity from $O^*(1.19^{|V_Q||V_G|})$ [23] to $O^*(|V_Q|^{(|V_G|+1)})$ [21], and to $O^*((|V_Q|+1)^{|V_G|})$ [41], which is our best-known worst-case time complexity for the problem. Here, O^* suppresses the 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 costs in practice. On the other hand, quite a few algorithms have been

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

[1] Not to be confused with graph embeddings.

developed towards improving the practical performance [23, 25–29, 43, 48]. They are all backtracking (also known as branch-and-bound) algorithms, among which the recent works [25, 26, 48] are based on a newly proposed backtracking framework called McSplit [28]. 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 (i.e., the time overhead for exploring those branches that do not lead to finding the maximum 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 theoretical guarantee on the worst-case time complexity.

Our method. In this paper, we develop an efficient backtracking algorithm called RRSplit, which leverages newly-designed reductions and upper bounds for decreasing the redundant computations. RRSplit achieves a worst-case time complexity of $O^*((|V_G|+1)^{|V_Q|})$, matching our best-known worst-case time complexity for the problem [41]. We note that this theoretical result is remarkable since (1) the algorithm proposed in [41] is of theoretical interest only and is not practically efficient and (2) McSplit and its variants do not have any theoretical guarantee on the worst-case time complexity. Specifically, RRSplit combines the following two kinds of reductions, namely vertex-equivalence-based reductions and maximality-based reductions, and the vertex-equivalence-based upper bound. We remark that the proposed reductions and upper bound are orthogonal to the existing upper bound techniques.

Vertex-equivalence-based reductions reduce the redundant computations induced by *common subgraph isomorphism (cs-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 equivalently, g is isomorphic to g'). All cs-isiomorphic 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. 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 [30] and an *auxiliary data structure* for verifying Condition 1 and Condition 2, respectively (details will be presented in Section 4.1).

Maximality-based reductions capture the redundant computations induced by *maximality*. In specific, 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 will incur redundant computations. To reduce them, we observe one necessary condition for a branch to hold the largest common subgraphs (details will be presented 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 [28] and thus can help to prune more branches (details will be presented in Section 4.3).

Contributions. We summarize our contributions as follows.

- We introduce the vertex-equivalence-based reductions for reducing the redundant computation induced by cs-isomorphism in Section 4.1. We further propose the maximality-based reductions for pruning those branches that hold non-maximal common subgraph only in Section 4.2. Finally, we develop a new vertex-equivalence-based upper bound for pruning more branches in Section 4.3.
- We propose a new backtracking algorithm called RRSplit, which is based on the newly-designed reductions. It has the 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 four 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 sate-of-the-art variant McSplitDAL. In Section 6 we discuss related work and conclude the paper in Section 7.

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 the graphs without vertex labels, but our methods can be easily adapted to vertex-labeled graphs. Consider two graphs $Q = (V_Q, E_Q)$ and $G = (V_G, E_G)$, with vertex sets V_Q and V_G and edge sets E_Q and 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 two simple graphs without labels.

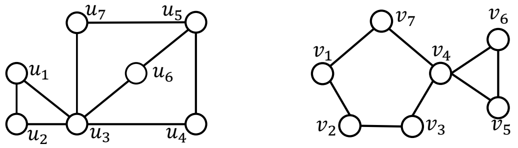
Definition 1 (Graph isomorphism [29]). Q is said to be isomorphic to G if and only if there exists a **bijection** $\phi : V_Q \to V_G$ such that

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

Based on the above definition, two isomorphic graphs are structurally equivalent, and thus we have $|V_Q| = |V_G|$ and $|E_Q| = |E_G|$. We then review the induced subgraph isomorphism for two graphs.

Definition 2 (Induced subgraph isomorphism [29]). Q is said to be (induced) subgraph isomorphic to G if and only if there exists an **injection** $\phi : V_Q \to V_G$ such that

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



(a) Input graph Q

(b) Input graph G

Figure 1: Input graphs used throughout the paper

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 $|E_Q| \leq |E_G|$; and 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 [29]). 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 ϕ, such that q is isomorphic to g under the bijection $\phi : V_q \rightarrow V_g$.

We refer to the size of a common subgraph $\langle q, g, \phi \rangle$ as its cardinality, i.e., number of vertices in q or g. Clearly, we have the size of a common subgraph at most $\min\{|V_Q|, |V_G|\}$. Besides, for the ease of presentation, we represent a common subgraph $\langle q, g, \phi \rangle$ by a set of vertex pairs $\{\langle u, \phi(u) \rangle \mid \forall u \in V_q\}$ in 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* $\phi := \{u_1 \rightarrow v_5, u_2 \rightarrow v_6, u_3 \rightarrow v_4, u_4 \rightarrow v_3, u_7 \rightarrow v_7\}$.

Below, we are ready to formulate the problem studied in this paper.

Problem 1 (Maximum Common Subgraph [24]). *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 the MCS problem is NP-hard [24] and is hard to approximate, i.e., there is no r-approximate PTIME algorithm for the problem for any $r \geq 1$ [18].

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, 25, 26, 48]. 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., $V_q \times V_g = \{\langle u, v \rangle \mid u \in V_q, v \in V_g\}$. Clearly, 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 $(S \cup \{\langle u, v \rangle\}, C \backslash u \backslash v)$, where $C \backslash u \backslash v$ denotes the set obtained by removing from C all candidate pairs containing u or v. **In the second group**, we form only one branch by excluding from C all candidate pairs containing u, i.e., $(S, C \backslash u)$. 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 D terms in the figure should be ignored for this example). The splitting process is partially depicted in Figure 2. For the initial branch* $B_0 = (\emptyset, \{u_1, u_2, ..., u_7\} \times \{v_1, v_2, ..., v_7\})$, *McSplit selects the branching vertex* u_1, *and then creates the first group of branches* $B_i = (\{\langle u_1, v_i \rangle\}, \{u_2, ..., u_7\} \times (\{v_1, v_2, ..., v_7\} \backslash \{v_i\}))$ *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* $B_8 = (\emptyset, \{u_2, ..., u_7\} \times \{v_1, v_2, ..., v_7\})$, *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 current found common subgraph (details will be discussed in Section 4.3). Below, we elaborate a bit more on 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 can be verified based on Definition 3. We note that the above reduction rule can be conducted in a *recursive* way. More precisely, for 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. Consider an immediate sub-branch of B_0 which is formed

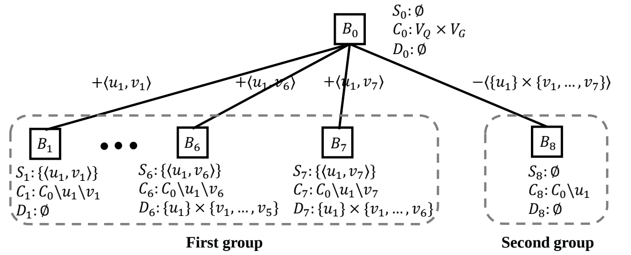

B_0
S_0: ∅
C_0: $V_Q \times V_G$
D_0: ∅
$+\langle u_1, v_1\rangle$
$+\langle u_1, v_6\rangle$
$+\langle u_1, v_7\rangle$
$-\langle\{u_1\} \times \{v_1, ..., v_7\}\rangle$
B_1
B_6
B_7
B_8
S_1: $\{\langle u_1, v_1\rangle\}$
C_1: $C_0\backslash u_1\backslash v_1$
D_1: ∅
S_6: $\{\langle u_1, v_6\rangle\}$
C_6: $C_0\backslash u_1\backslash v_6$
D_6: $\{u_1\} \times \{v_1, ..., v_5\}$
S_7: $\{\langle u_1, v_7\rangle\}$
C_7: $C_0\backslash u_1\backslash v_7$
D_7: $\{u_1\} \times \{v_1, ..., v_6\}$
S_8: ∅
C_8: $C_0\backslash u_1$
D_8: ∅
First group
Second group

Figure 2: Illustrating the backtracking process ("+" means to move vertex pairs from C to S and "-" means to remove vertex pairs from C)

by including candidate pair $\langle u, v\rangle$ to the partial solution. Those candidate pairs in $N(u, V_Q)\times\overline{N}(v, V_G)$ and $\overline{N}(u, V_Q)\times N(v, V_G)$ can be pruned by the reduction rule. As a result, the refined candidate set is $N(u, V_Q)\times N(v, V_G)\cup\overline{N}(u, V_Q\backslash\{u\})\times\overline{N}(v, V_G\backslash\{v\})$, which is *split* as two subsets. In general, for a branch (S, C), the refined candidate set C consists of at most $2^{|S|}$ subsets, i.e., $C = X_1 \times Y_1 \cup \cdots \cup X_c \times Y_c$ where $1 \leq c \leq 2^{|S|}$. For a sub-branch of (S, C) which is formed by moving a candidate pair $\langle u, v\rangle$ from C to S, the refined candidate set is

$$\bigcup_{i=1}^{c} N(u, X_i) \times N(v, Y_i) \cup \overline{N}(u, X_i\backslash\{u\}) \times \overline{N}(v, Y_i\backslash\{v\}) \qquad (3)$$

Example 3. *Consider branch B_6 in the first group in Figure 2. We have $X_1 = N(u_1, V_Q) = \{u_2, u_3\}$, $X_2 = \overline{N}(u_1, V_Q\backslash\{u_1\}) = \{u_4, u_5, u_6, u_7\}$, $Y_1 = \{v_4, v_5\}$ 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 $\{u_5\} \times \{v_1\} \cup \{u_4, u_6\} \times \{v_2, v_3\}$.*

Summary. 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 the upper-bound-based pruning (Line 5). Besides, it branches by selecting 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 to 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 [25, 26, 48]. 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.

Algorithm 1: An existing framework: McSplit [28]

```
    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
          $C$ based on a policy;
7       $Y_{temp} \leftarrow Y$;
        /* First group: branches formed by including $u$ */
8       for $i = 1, 2, ..., |Y|$ 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 (3);
11          McSplit-Rec($S \cup \{\langle u, v\rangle\}, C_i$);
        /* Second group: one branch formed by excluding $u$ */
12      McSplit-Rec($S, C\backslash u$);
```

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 of the size of common subgraphs to 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 the worst-case time complexity of $O^*((|V_G|+1)^{|V_Q|})$, matching our 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 a new concept called *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 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. We reduce this redundancy as follows. For a common subgraph $\langle q, g, \phi\rangle$ to be found in a branch, if there exists another one that is cs-isomorphic to $\langle q, g, \phi\rangle$ (Condition 1) and has been found before (Condition 2), we can safely ignore the common subgraph

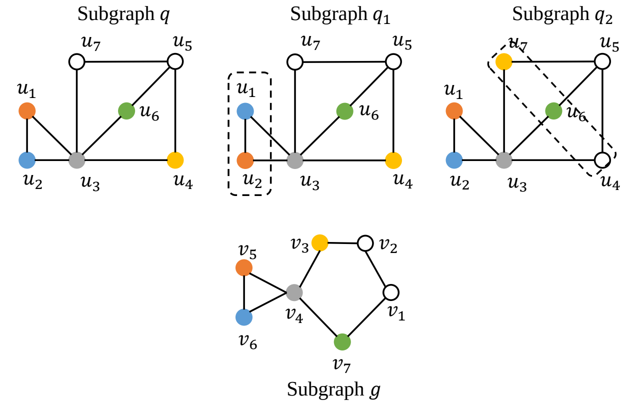


Figure 3: Illustrating cs-isomorphism and vertex equivalence (vertices, denoted by colored bullet circles, induce subgraphs q, q_1, q_2 and g; vertices in $\{u_1, u_2\}$ and $\{u_4, u_6, u_7\}$ are structural equivalent, respectively; $\langle q, g, \phi\rangle$ is cs-isomorphic to $\langle q_1, g, \phi_1\rangle$ (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)

$\langle q, g, \phi\rangle$. To facilitate the reduction, we will leverage the *vertex equivalence* property and an *auxiliary data structure* for verifying Condition 1 and Condition 2, respectively.

Vertex equivalence. The *structural equivalence* property has been widely used to speed up subgraph matching tasks [19, 30, 46]. Conceptually, two vertices are *structurally equivalent* if and only if they have the *same* set of neighbours. We provide the formal definition below.

Definition 5 (Structural equivalence [30]). Two vertices u and v in Q are structurally equivalent, denoted $u \sim v$, if and only if

$$\forall u' \in V_Q, (u, u') \in E(Q) \Leftrightarrow (v, u') \in E(Q). \tag{4}$$

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

$$\Psi(u) := \{u' \in V_Q \mid u' \sim u\}, \tag{5}$$

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 [19, 30, 46].

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 structural equivalent counterpart, which fall in two cases. In specific, consider a vertex u in V_q and one of its structural equivalent counterpart 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 $\langle u', \phi(u)\rangle$ with $\langle u, \phi(u)\rangle$ and $\langle u', \phi(u')\rangle$; Otherwise, we replace u_{equ} with u, 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).

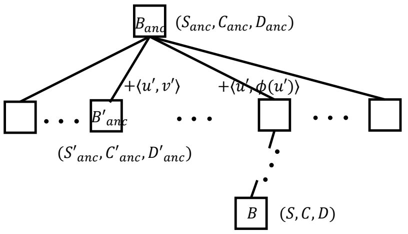

B_{anc}
$(S_{anc}, C_{anc}, D_{anc})$
$+\langle u', v'\rangle$
$+\langle u', \phi(u')\rangle$
B'_{anc}
$(S'_{anc}, C'_{anc}, D'_{anc})$
B
(S, C, D)

Figure 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)

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 either of the following case holds.*

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

Case 2: *$u' \notin V_q$. $S' = S\backslash\{\langle u, \phi(u)\rangle\} \cup \{\langle u', \phi(u)\rangle\}$ 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).

- **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 $D_i = D \cup \{\langle u, v_1\rangle, \langle u, v_2\rangle, \cdots, \langle u, v_{i-1}\rangle\}$. 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, 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 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 structural equivalent to u, i.e., $u_{equ} \in \Psi(u)$*, Condition 1&2 holds for S_{sub} and thus the branch can be pruned. Below, we elaborate on the details.

We first show that Condition 1 holds as follows. 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).

$$S_{iso} = S_{sub}\backslash\{\langle u, v\rangle, \langle u_{equ}, \phi(u_{equ})\rangle\} \cup \{\langle u_{equ}, v\rangle, \langle u, \phi(u_{equ})\rangle\} \quad (6)$$

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

We then 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) $S_{iso}\backslash\{\langle u, \phi(u_{equ})\rangle\}$ 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 (6) has been found before the formation of (S, C, D).*

Proof. We put the details in the technical report. □

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 where u_2 is selected as the branching vertex. We can see that u_1 is in $\Psi(u_2)$ and $D_6 = \{u_1\} \times \{v_1, v_2, v_3, v_4, v_5\}$. Thus, for two sub-branches of B_6 formed by including $\langle u_2, v_4\rangle$ or $\langle u_2, v_5\rangle$, we can prune them based on the above reduction.*

Reduction at the second group. Consider the sub-branch formed 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 $C\backslash u$ and is structural 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).

$$S_{iso} = S_{sub}\backslash\{\langle u_{equ}, \phi_{sub}(u_{equ})\rangle\} \cup \{\langle u, \phi_{sub}(u_{equ})\rangle\} \quad (7)$$

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) $\langle u, \phi_{sub}(u_{equ})\rangle$ 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.

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 (7) has been found before the formation of $(S, C\backslash u, D)$ at the second group.*

Proof. We put the details in the technical report. □

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

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 $(V_Q\backslash\{u_1, u_2\}) \times V_G$, i.e., $\{u_3, u_4, u_5, u_6, u_7\} \times \{v_1, v_2, \cdots, v_7\}$.*

4.2 Maximality-based Reduction

We introduce the redundancies induced by *non-maximality*. In specific, 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 $B = (S, C, D)$. We observe that *there exists one largest common subgraph in B that must contain some specific candidate vertex pairs.* Note that there might exist multiple common subgraphs with the largest number of vertices. As a result, we can safely prune all other sub-branches that have all common subgraphs inside *exclude* these candidate vertex pairs. 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 C, u and v are simultaneously adjacent or non-adjacent to all other vertices in X and Y, respectively, i.e.,

$$\forall X \times Y \in C : N(u, X) = X\backslash\{u\}, N(v, Y) = Y\backslash\{v\} \text{ or } N(u, X) = \emptyset, N(v, Y) = \emptyset, \quad (8)$$

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 (8). 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.

Case 1: $u \notin V_{q^*}$ and $v \in V_{g^*}$. In this case, there exists a vertex pair $\langle \phi^{*-1}(v), v\rangle$ in S^* where ϕ^{*-1} is the inverse of ϕ^*. We construct

S_{opt} by replacing the vertex pair $\langle \phi^{*-1}(v), v\rangle$ with $\langle u, v\rangle$, i.e.,

$$S_{opt} = S^{*}\backslash\{\langle \phi^{*-1}(v), v\rangle\} \cup \{\langle u, v\rangle\}. \tag{9}$$

Clearly, we have $S \subseteq S_{opt} \subseteq S \cup C$ (i.e., S_{opt} is in B) since S^* is in B and $\langle u, v\rangle$ is in the candidate set C. Besides, we have $|S_{opt}| = |S^*|$ and $\langle u, v\rangle \in S_{opt}$ based on the above construction. Finally, we deduce that S_{opt} is a common subgraph by showing that any two vertex pairs in S_{opt} satisfy Equation (1), i.e., g_{opt} is isomorphic to q_{opt} under the bijection ϕ_{opt}. First, $S^{*}\backslash\{\langle \phi^{*-1}(v), v\rangle\}$, as a subset of S^*, is a common subgraph and thus has any two vertex pairs inside satisfying Equation (1) (note that any subset of a common subgraph is still a common subgraph); Second, for each pair $\langle u', v'\rangle$ in S, u is adjacent to u' if and only if v is adjacent to v' (since $\langle u, v\rangle$ is a candidate pair which can form a common subgraph with S); Third, for each pair $\langle u', v'\rangle$ in $S_{opt}\backslash S\backslash\{\langle \phi^{*-1}(v), v\rangle\}$, it is clear that $\langle u', v'\rangle$ is in one subset $X \times Y$ of C and thus u is adjacent to u' if and only if v is adjacent to v' based on Equation (8). Therefore, any two vertex pairs in S_{opt} will satisfy the Equation (1).

Case 2: $u \in V_{q^*}$ and $v \notin V_{g^*}$. There exists a vertex pair $\langle u, \phi^*(u)\rangle$ in S^*. We construct S_{opt} by replacing $\langle u, \phi^*(u)\rangle$ with $\langle u, v\rangle$, i.e., $S_{opt} = S^{*}\backslash\{\langle u, \phi^*(u)\rangle\} \cup \{\langle u, v\rangle\}$. Similar to Case 1, we can prove that S_{opt} includes $\langle u, v\rangle$ and is one largest common subgraph to be found in B.

Case 3: $u \in V_{q^*}$ and $v \in V_{g^*}$. There exists two distinct vertex pairs $\langle u, \phi^*(u)\rangle$ and $\langle \phi^{*-1}(v), v\rangle$ in S^*. We construct S_{opt} by replacing these two vertex pairs with $\langle \phi^{*-1}(v), \phi(u)\rangle$ and $\langle u, v\rangle$, formally,

$$S_{opt}{=}S^{*}\backslash\{\langle u, \phi^*(u)\rangle, \langle \phi^{*-1}(v), v\rangle\}{\cup}\{\langle \phi^{*-1}(v), \phi^*(u)\rangle, \langle u, v\rangle\}. \tag{10}$$

Clearly, we have $S \subseteq S_{opt} \subseteq S{\cup}C$ (i.e., S_{opt} is in B), $|S_{opt}| = |S^*|$ and $\langle u, v\rangle \in S_{opt}$ based on the above construction. We then deduce that S_{opt} is a common subgraph by showing that any two vertex pairs in S_{opt} satisfy Equation (1). First, $S^{*}\backslash\{\langle u, \phi^*(u)\rangle, \langle \phi^{*-1}(v), v\rangle\}$, as a subset of S^*, is a common subgraph and thus has any two vertex pairs inside satisfying Equation (1); Second, consider a vertex pair $\langle u', v'\rangle$ in $S^{*}\backslash\{\langle u, \phi^*(u)\rangle, \langle \phi^{*-1}(v), v\rangle\}$. Similar to Case 1, we can prove that u is adjacent to u' if and only if v is adjacent to v'. Besides, we show that $\phi^{*-1}(v)$ is adjacent to u' if and only if $\phi(u)$ is adjacent to v' since (1) $(\phi^{*-1}(v), u') \in E_Q \Leftrightarrow (v, v') \in E_G$ and $(u, u') \in E_Q \Leftrightarrow (\phi^*(u), v') \in E_G$ (since the common subgraph S^* contains $\{\langle u, \phi^*(u)\rangle, \langle \phi^{*-1}(v), v\rangle\}$), (2) $(v, v') \in E_G \Leftrightarrow (u, u') \in E_Q$ (as we shown above), and thus (3) they can be combined as $(\phi^{*-1}(v), u') \in E_Q \Leftrightarrow (v, v') \in E_G \Leftrightarrow (u, u') \in E_Q \Leftrightarrow (\phi^*(u), v') \in E_G$. Third, we have $(u, \phi^{*-1}(v)) \in E_Q \Leftrightarrow (v, \phi^*(u)) \in E_G$ since the common subgraph S^* contains $\{\langle u, \phi^*(u)\rangle, \langle \phi^{*-1}(v), v\rangle\}$ and thus $(u, \phi^{*-1}(v)) \in E_Q \Leftrightarrow (\phi^*(u), v) \in E_G$ (note that $(\phi^*(u), v)$ refers to the same edge as $(v, \phi^*(u))$ since the graphs Q and G are undirected). Therefore, any two vertex pairs in R_{opt} will satisfy Equation (1).

Case 4: $u \notin V_{q^*}$ and $v \notin V_{g^*}$. We note that this case will not occur since otherwise the contradiction is derived by showing that $S^* \cup \{\langle u, v\rangle\}$ is a larger common subgraph (note that the proof is similar to Case 1 and thus be omitted). □

Consider a branch $B = (S, C, D)$ where $X^* \times Y^*$ in 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 (8). 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 $(S \cup \{\langle u^*, v\rangle\}, C\backslash u^*\backslash v, D \cup \{u^*\} \times (Y^*\backslash\{v\}))$ 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 $D \cup \{u^*\} \times (Y^*\backslash\{v\})$ 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 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 (8), only one sub-branch $(S{\cup}\{\langle u, v\rangle\}, C\backslash u\backslash v, D{\cup}\{u\}{\times}(Y\backslash\{v\}))$ 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 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 (8) since (1) $N(u_2, X_1) = X_1\backslash\{u_2\}$ and $N(v_5, Y_1) = Y_1\backslash\{v_5\}$ and (2) $N(u_2, X_2) = \emptyset$ and $N(v_5, Y_2) = \emptyset$. Therefore, we only need to explore one sub-branch $(S_6 \cup \{\langle u_2, v_5\rangle\}, C_6\backslash u_2\backslash v_5, D_6 \cup \{\langle u_2, v_4\rangle\})$, 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 of 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 is, 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 C, we can derive

$$|S_{sub}| \cap X \times Y \le ub_{X,Y} := \min\{|X|, |Y|\} \tag{11}$$

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 C are disjoint, the following existing upper bound of branch (S, C, D), denoted by $ub_{S,C}$ [28], can be derived.

$$|S_{sub}| \le ub_{S,C} := |S| + \sum_{X \times Y \in C} ub_{X,Y} \tag{12}$$

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 $\langle X, Y \rangle$ of C. Let u be an arbitrary vertex in X. We partition X and Y as below.

$$X_L = X \cap \Psi(u), X_R = X \backslash X_L \tag{13}$$

$$Y_L = \{v \mid \langle u', v \rangle \in D, u' \in \Psi(u)\}, Y_R = Y \backslash Y_L, \tag{14}$$

where X_L consists of those vertices in X that are structural equivalent to u and Y_L consists of those vertices in Y each of which v appears in a vertex pair $\langle u', v \rangle$ in D such that $u' \in \Psi(u)$. 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

$$ub_{X,Y,D} := \min\{|X_R|, |Y|\} + \min\{|X_L|, |Y_R|, \max\{|Y| - |X_R|, 0\}\}. \tag{15}$$

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

$$|S_{sub}| \leq ub_{S,C,D} := |S| + \sum_{X \times Y \in C} ub_{X,Y,D}. \tag{16}$$

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., $ub_{S,C,D} \leq ub_{S,C}$.

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 $D_6 = \{u_1\} \times \{v_1, v_2, ..., v_5\}$. 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 $ub_{S,C} = 7$.*

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. First, it maintains one additional auxiliary data structure, namely exclusion set D, for each formed branch, which is initialized as an empty set and recursively updated as discussed. Second, 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 one 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. Third, it creates only one sub-branch and prunes all others if the maximality-based reduction is triggered (Lines 9-11). Forth, 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 $S^* \leftarrow S$;
       /* Termination (Lemma 5)                          */
6      if $C = \emptyset$ then return;
7      if $ub_{S,C,D} \leq |S^*|$ then return;
       /* Branching                                      */
8      Select a branching vertex $u$ and a branching subset $X \times Y$ from
         $C$ based on a policy;
       /* Maximality-based reduction                     */
9      if there exists a vertex $v$ in $Y$ such that $\langle u, v \rangle$ satisfies
         Equation (8) then
10         RRSplit-Rec($S \cup \{\langle u, v \rangle\}, C \backslash u \backslash v, D \cup \{u\} \times (Y \backslash \{v\})$);
11         return;
       /* Branching at the first group                   */
12     $Y_{temp} \leftarrow Y$;
13     for $i = 1, 2, ..., |Y|$ do
14         Select and remove a vertex $v$ from $Y_{temp}$ based on a policy;
15         if there exists a vertex pair $\langle u', v \rangle$ in $D$ such that
             $u' \in \Psi(u)$ then
16             continue;
17         Refine candidate set $C \backslash u \backslash v$ as $C_i$ based on Equation (3);
18         RRSplit-Rec($S \cup \{\langle u, v \rangle\}, C_i, D \cup \{u\} \times (Y \backslash Y_{temp})$);
       /* Branching at the second group                  */
19     RRSplit-Rec($S, C \backslash \Psi(u), D$);
```

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 two 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., $C = X_1 \times Y_1 \cup X_2 \times Y_2 \cup \cdots \cup X_c \times Y_c$ where c is a positive integer, based on Equation (3). We note that subsets in $X_1, X_2, ..., X_c$ (resp. $Y_1, Y_2, ..., Y_c$) are mutually disjoint and $X_1 \cup X_2 \cup .... \cup X_c = X$ (resp. $Y_1 \cup Y_2 \cup .... \cup Y_c = Y$), as discussed in the proof of Lemma 2. Therefore, C can be stored as c subsets, each of which $\langle X_i, Y_i\rangle$ ($1 \le i \le c$) 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., $D = \{u_1\} \times A_1 \cup \{u_2\} \times A_2 \cup \cdots \cup \{u_d\} \times A_d$ 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). As a result, we have $\sum_{u_i \in \Psi(u)}(|A_i|) \le |V_G|$. Second, the reduction at the second group runs in $O(|X|)$ for updating $C\backslash\Psi(u)$ 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 structural 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 (8). Forth, 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 $\sum_{X' \times Y' \in C}(|X'| + |V_G|)$, 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 $|V_Q| \le |V_G|$. 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 $u \ne u'$ and $v \ne v'$. Based on this, let $|S| = k$ where $0 \le k \le |V_Q|$, 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

$$T = |V_Q| \sum_{k=0}^{|V_Q|} k! \binom{|V_Q|}{k}\binom{|V_G|}{k}. \tag{17}$$

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

$$T = |V_Q| \sum_{k=0}^{|V_Q|} (|V_Q| - k)! \binom{|V_Q|}{k}\binom{|V_G|}{|V_Q| - k} \tag{18}$$

$$= |V_Q| \sum_{k=0}^{|V_Q|} \frac{(|V_G|)!}{(|V_G| - |V_Q| + k)!}\binom{|V_Q|}{k} \tag{19}$$

$$\le |V_Q| \sum_{k=0}^{|V_Q|} (|V_G|)^{|V_Q| - k}\binom{|V_Q|}{k} = |V_Q|(|V_G| + 1)^{|V_Q|}, \tag{20}$$

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. □

Remark. We remark that the achieved worst-case time complexity $O^*((|V_G|+1)^{|V_Q|})$ of RRSplit is the same as, to our best knowledge, our best-known worst-case time complexity for the problem [41]. However, the algorithm proposed in [41] is of theoretical interest only and is not practically efficient. Besides, we note that McSplit and its variants [25, 26, 28, 48] do not have any theoretical guarantee on the worst-case time complexity.

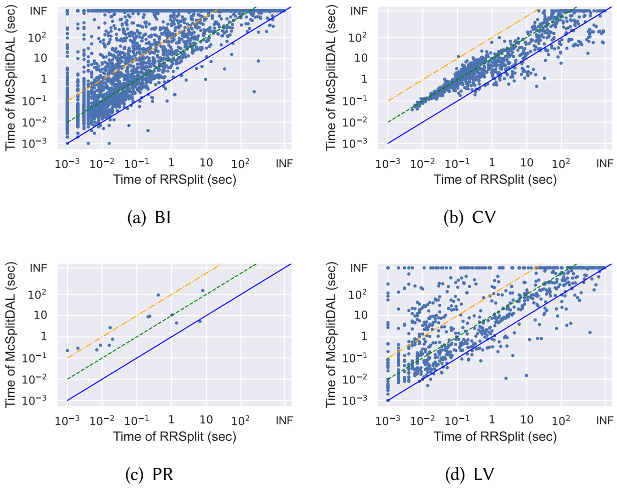
5 EXPERIMENTS

Datasets. We use four benchmark graph collections, namely biochemicalReactions (BI), images-CVIU11 (CV), images-PR15 (PR) and LV (LV), in the experiments, which have been widely used in existing studies [16, 25, 26, 28, 36, 48]. 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. Following existing studies [16, 25, 26, 28, 36, 48], 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, we test all those problem instances with one graph Q from pattern graphs and the other G from target graphs.

Algorithms. We compare the newly proposed algorithm RRSplit with McSplitDAL [26]. 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 [48] and McSplitRL [26]. Besides, we evaluate two variants of our algorithm RRSplit, namely RRSplit-MR and RRSplit-VER, to study the effectiveness of different reductions employed in RRSplit.

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%



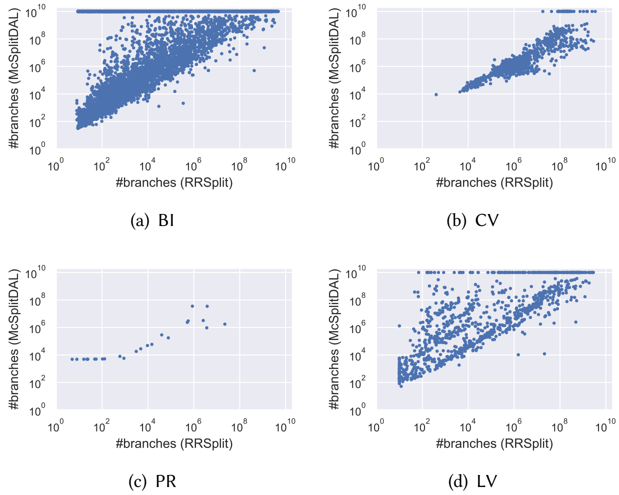
(a) BI

(b) CV

(c) PR

(d) LV

Figure 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× (resp. 10×) speedup compared with `McSplitDAL`.



(a) BI

(b) CV

(c) PR

(d) LV

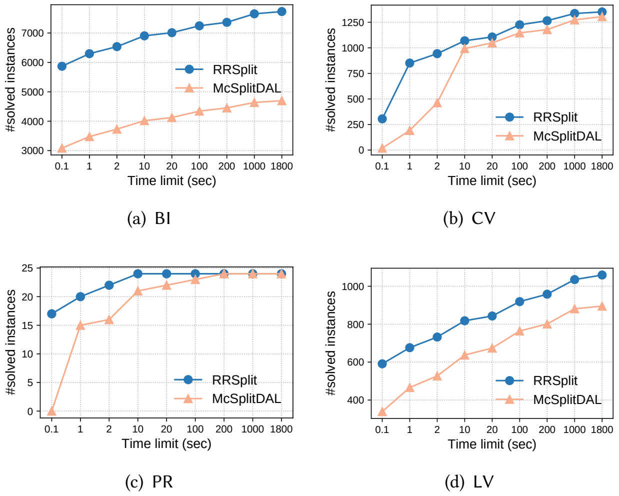
Figure 6: Number of formed branches on all datasets

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 [26]. 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 XXXXXX.

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 [27], 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`. Besides, we mark the running time as INF if the problem instance cannot be solved within the default time limit. We observe that (1) `RRSplit` outperforms `McSplitDAL` by achieving around one to three orders of magnitude speedup on the majority of the tested problem instances and (2) `McSplitDAL` cannot handle all problem instances within the time limit. In particular, we note that `McSplitDAL` runs slightly faster on a few 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 polices 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 (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 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.



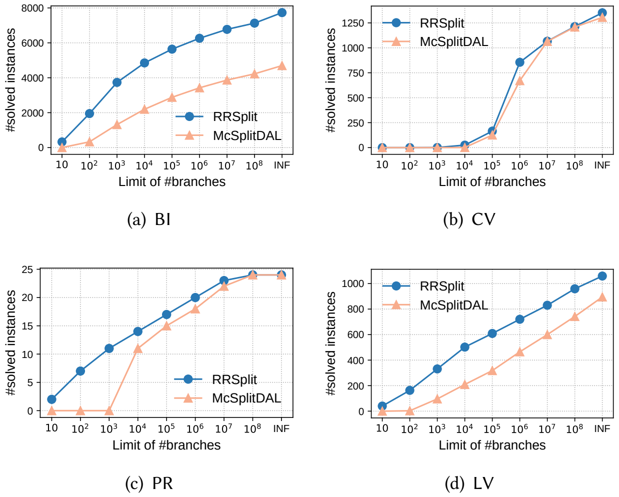
(a) BI

(b) CV

(c) PR

(d) LV

Figure 7: Comparison by varying time limits



(a) BI

(b) CV

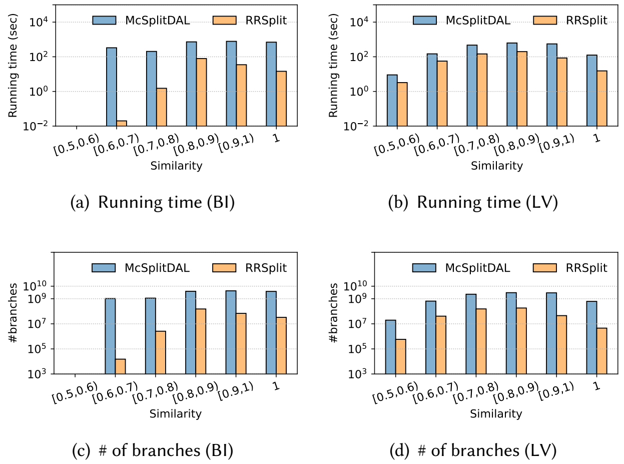
(c) PR

(d) LV

Figure 8: Comparison by varying the limit of number of formed branches

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 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 of the running time attribute to our newly-designed reductions.

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



(a) Running time (BI)

(b) Running time (LV)

(c) # of branches (BI)

(d) # of branches (LV)

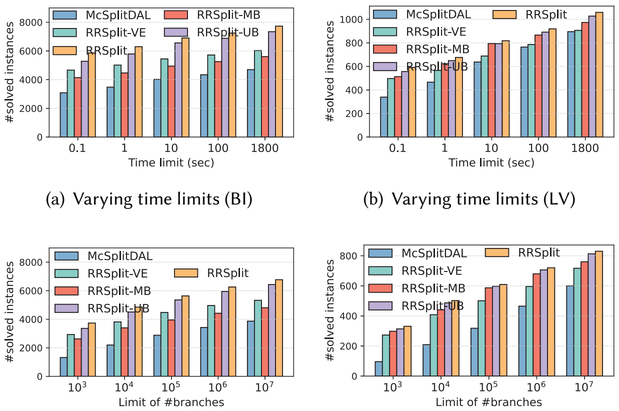
Figure 9: Comparison by varying similarities

Varying the limits of number of formed branches. We report the number of solved problem instances in Figure 8 as the limit of number of formed branches varies. 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 the similar tendencies as those in Figure 7 in general. This further shows the practical superiority of the newly proposed reductions.

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

$$Sim(Q,G) = \frac{|S^*|}{\min\{|V_Q|,|V_G|\}}, \tag{21}$$

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 Figure 9(a) and (b) and the average number of formed branches in Figure 9(c) and (d). The results on CV and PR show similar clues, which we put in the technical report []. 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. Possible reasons include (1) the number of common subgraphs (i.e.,



(a) Varying time limits (BI)

(b) Varying time limits (LV)

(c) Varying limit of #branches (BI)

(d) Varying limit of #branches (LV)

Figure 10: Comparison among various reductions

search space) first increases and then decreases as the similarity grows and/or (2) the proposed reductions performs better on those problem instances with the similarity close to 0.5 or 1.

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 and PR show similar clues, which we put in the technical report []. 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. The possible reasons include that BI and LV come from different domains and have different properties.

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, 21, 23, 25–29, 41, 43, 48] or approximately [4, 10, 33, 44, 47]. 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 [23, 29]. Specifically, authors in [23, 29] 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 [27, 43] 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. [28] propose a new backtracking framework, namely McSplit, which is not based on the maximum clique search problem. Recent works [25, 26, 48] follow McSplit and improve the practical performance by optimizing the policies of branching via learning techniques. Among them, McSplitDAL [26] runs faster than others. We note that some exact algorithms are designed to achieve improvements of theoretical time complexity [1, 21, 23, 41]. They have gradually improved the worst-case time complexity from $O^*(1.19^{|V_Q||V_G|})$ [23] to $O^*(|V_Q|^{(|V_G|+1)})$ [21], and to $O^*((|V_Q|+1)^{|V_G|})$ [41], 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 polices proposed in [25, 26, 48] 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 [10, 33], spectra methods [44], and learning-based methods [4, 47]. 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, 11, 13, 14, 17, 20, 35, 37–39, 42]. 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, 13, 14, 20]. 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 other types of graphs, including vertex-labeled and edge-labeled graphs.

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-Velar, 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. Glsearch: 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] 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.

[10] Jaeun 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Evgeny B Krissinel and Kim Henrick. 2004. Common subgraph isomorphism detection by backtracking search. *Software: Practice and Experience* 34, 6 (2004), 591–607.

[22] 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.

[23] Giorgio Levi. 1973. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo* 9, 4 (1973), 341–352.

[24] Harry R Lewis. 1983. Michael R. ΠGarey and David S. Johnson. Computers and intractability. A guide to the theory of NP-completeness. WH Freeman and Company, San Francisco1979, x+ 338 pp. *The Journal of Symbolic Logic* 48, 2 (1983), 498–500.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] James J McGregor. 1982. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience* 12, 1 (1982), 23–34.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] Xibo Sun and Qiong Luo. 2023. Efficient GPU-Accelerated Subgraph Matching. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.

[40] 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.

[41] 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: 11th Annual International Conference, COCOON 2005 Kunming, China, August 16–19, 2005 Proceedings 11*. Springer, 717–727.

[42] Julian R Ullmann. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)* 23, 1 (1976), 31–42.

[43] 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.

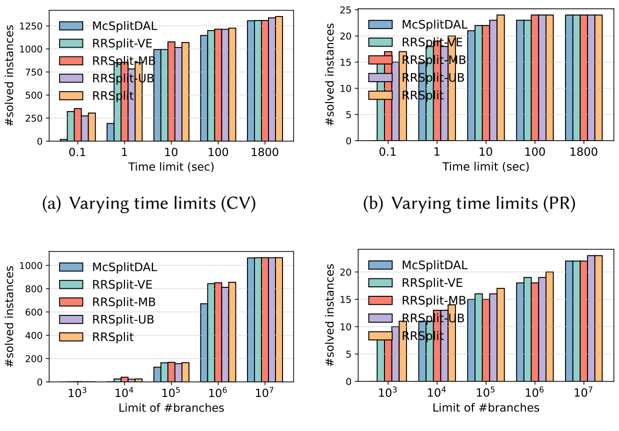
[44] Bai Xiao, Edwin R Hancock, and Richard C Wilson. 2009. A generative model for graph matching and embedding. *Computer Vision and Image Understanding* 113, 7 (2009), 777–789.

[45] 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.

[46] 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).

[47] 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.

[48] 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, Vienna, Austria, 23-29 July 2022*. 1908–1914.



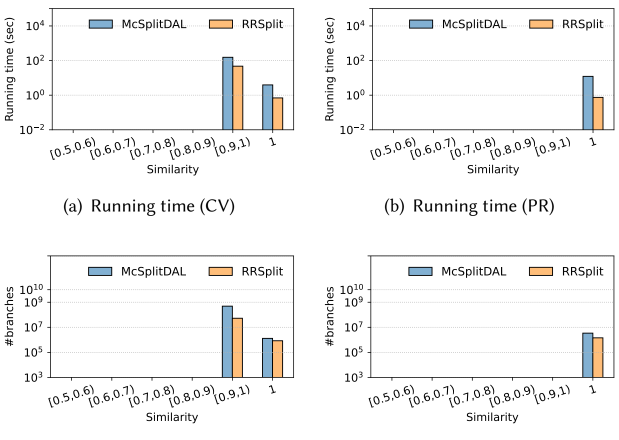
(a) Varying time limits (CV)

(b) Varying time limits (PR)

(c) Varying limit of #branches (CV)

(d) Varying limit of #branches (PR)

Figure 12: Comparison among various reductions (additional results)



(a) Running time (CV)

(b) Running time (PR)

(c) # of branches (CV)

(d) # of branches (PR)

Figure 11: Comparison by varying similarities (additional results)

A ADDITIONAL EXPERIMENTAL RESULTS

In this section, we provide additional experimental results, including *comparison by varying similarities* and *comparison among various reductions.*

Varying the similarities of two input graphs (additional results). We test different problem instances as the similarity varies on CV and PR. We remark that all tested problem instances in CV (resp. PR) have their similarities vary from 0.9 to 1 (resp. equal to 1). We report the average running time in Figure 11(a) and (b) and the average number of formed branches in Figure 11(c) and (d). The results show the similar clues to those on BI and LV. In specific, our RRSplit runs around 5×-10× faster and forms fewer branches than McSplitDAL.

Varying different reductions (additional results). We compare RRSplit with three variants, namely RRSplit-VE, RRSplit-MB and RRSplit-UB, on CV and PR. We report the number of solved problem instances in Figure 12 (a) and (b) for varying the time limit and in Figure 12 (c) and (d) for varying the limit of number of formed branches. The results on CV and PR show similar trends to those on BI and LV. First, we can see that all four algorithms performs better than McSplitDAL, among which RRSplit performs the best. This indicates the effectiveness of the proposed vertex-equivalence based reductions and maximality based reductions. Second, we note that RRSplit-MB and RRSplit-VE achieve the comparable performance.

B ADDITIONAL PROOFS

Lemma 2 *Let (S, C, D) be a branch. Common subgraph S_{iso} at Equation (6) 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 corresponding to a branch. Consider the path from the initial branch $(\emptyset, \langle V_Q \times V_G \rangle, \emptyset)$ 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} has 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 $(S \cup \{\langle u^*, v \rangle\}, C \backslash u^* \backslash v)$, and thus (4) by combining all the above, we have $S'_{asc} = S_{asc} \cup \{\langle u_{equ}, v \rangle\} \subseteq S_{sub} \cup \{\langle u_{equ}, v \rangle\} \backslash \{\langle u_{equ}, \phi(u_{equ}) \rangle, \langle u^*, v \rangle\} \subseteq S_{iso}$. Second, we have $S_{iso} \subseteq S'_{asc} \cup C'_{asc}$ based on the following two facts.

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

Fact 1 holds since (1) $S_{sub} \subseteq S_{asc} \cup C_{asc}$ (note that S_{sub} is a common subgraph in B_{asc}), (2) vertices u_{equ} and v do not appear in $S_{sub} \backslash \{\langle u_{equ}, \phi(u_{equ}) \rangle, \langle u^*, v \rangle\}$ and thus we can derive that $S_{sub} \backslash \{\langle u_{equ}, \phi(u_{equ}) \rangle, \langle u^*, v \rangle\} \subseteq (S_{asc} \cup C_{asc}) \backslash u_{equ} \backslash v$ (note that $\langle u_{equ}, \phi(u_{equ}) \rangle$ and $\langle u^*, v \rangle$ are the unique vertex pairs that consist of u_{equ} and v in S_{sub}, respectively), and (3) $S'_{asc} = S_{asc} \cup \{\langle u_{equ}, v \rangle\}$ and $C'_{asc} = C_{asc} \backslash u_{equ} \backslash v$ based on the branching rule.

Fact 2 can be verified as follows. Vertex pair $\langle u_{equ}, v \rangle$ is in S'_{asc} since $S'_{asc} = S_{asc} \cup \{\langle u_{equ}, v \rangle\}$. We note that vertices u_{equ}, $\phi(u_{equ})$, u^* and v appear in C_{asc} since $\langle u_{equ}, \phi(u_{equ}) \rangle$ and $\langle u^*, v \rangle$ are in C_{asc} discussed before. Let $C_{asc} = \langle X_1 \times Y_1 \rangle \cup \langle X_2 \times Y_2 \rangle \cup \cdots \cup \langle X_c \times Y_c \rangle$ where c is a positive integer. It is no hard to verify that, for two vertices u and u' (resp. v and v') appearing in C_{asc}, u and u' are in the same subset X_i (resp. Y_i) of C_{asc} with $1 \leq i \leq c$ if and only if u and u' (resp. v and v') have the same set of neighbours and non-neighbours in q (resp. g) according to Equation (3). Besides, we have $X_i \cap X_j = \emptyset$ and $Y_i \cap Y_j = \emptyset$ for $1 \leq i \neq j \leq c$ since each vertex appearing in the candidate set is split into exactly one subset according to Equation (3). Based on the above, we assume that u_{equ} appears in a subset $\langle X_i, Y_i \rangle$ of C_{asc} where u_{equ} is in X_i and $1 \leq i \leq c$. We can deduce that u^* is in X_i since u_{equ} and u^* are structural equivalent and thus they have the same set of neighbours and non-neighbours in q based on Definition **??**. Besides, we can deduce that vertices $\phi(u_{equ})$ and v are in Y_i since (1) $\langle u_{equ}, \phi(u_{equ})\rangle$ and $\langle u^*, v\rangle$ are in C_{asc}, (2) u^* and v are in exactly one subset X_i, and thus (3) they must appear in $\langle X_i \times Y_i\rangle$. □

Lemma 3 *Let (S, C, D) be a branch where u^* is selected as the branching vertex. Common subgraph S_{iso} at Equation (7) has been found before the formation of $(S, C\backslash u^*, D)$ at the second group.*

Proof. First, we note that $\langle u_{equ}, \phi_{iso}(u_{equ})\rangle$ is in $C\backslash u^*$ and also in C since otherwise S_{sub} cannot include $\langle u_{equ}, \phi_{iso}(u_{equ})\rangle$. This is because (1) $S \subseteq S_{sub} \subseteq S \cup C\backslash u^*$ since S_{sub} is a common subgraph in the sub-branch $(S, C\backslash u^*, D)$ and (2) S does not include $\langle u_{equ}, \phi_{iso}(u_{equ})\rangle$ since u_{equ} appears in $C\backslash u^*$. Second, we note that $\phi_{iso}(u_{equ})$ is in Y^*. Recall that $\langle X^* \times Y^*\rangle$ 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 structural 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 $(S \cup \{\langle u^*, \phi_{iso}(u_{equ})\rangle\}, C\backslash u^*\backslash\phi_{iso}(u_{equ}), D')$, which is formed at branch (S, C, D) by including $\langle u^*, \phi_{iso}(u_{equ})\rangle$ before the formation of $(S, C\backslash u^*, D)$, since $\phi_{iso}(u_{equ}) \in Y^*$. Forth, we show that S_{iso} is in $(S \cup \{\langle u^*, \phi_{iso}(u_{equ})\rangle\}, C\backslash u^*\backslash\phi_{iso}(u_{equ}), D')$, formally, $S \cup \{\langle u^*, \phi_{iso}(u_{equ})\rangle\} \subseteq S_{iso} \subseteq S \cup \{\langle u^*, \phi_{iso}(u_{equ})\rangle\} \cup (C\backslash u^*\backslash\phi_{iso}(u_{equ}))$. We have $S \subseteq S_{sub} \subseteq S \cup (C\backslash u^*)$ since S_{sub} is a common subgraph in $(S, C\backslash u^*, D)$. Let $S' = S \cup \{\langle u^*, \phi_{iso}(u_{equ})\rangle\}$, it can be proved as below.

$$S \subseteq S_{sub} \subseteq S \cup (C\backslash u^*) \tag{22}$$

$$\Rightarrow S \subseteq S_{sub}\backslash\{\langle u_{equ}, \phi_{iso}(u_{equ})\rangle\} \subseteq S \cup (C\backslash u^*\backslash\phi_{iso}(u_{equ})) \tag{23}$$

$$\Rightarrow S' \subseteq S_{iso} \subseteq S' \cup (C\backslash u^*\backslash\phi_{iso}(u_{equ})) \tag{24}$$

Note that Equation (23) holds since $\langle u_{equ}, \phi_{iso}(u_{equ})\rangle$ is in S; Equation (24) is derived by including the vertex pair $\langle u^*, \phi_{iso}(u_{equ})\rangle$. □