PathLAD+: An Improved Exact Algorithm for Subgraph Isomorphism Problem

Yiyuan Wang[1,2], **Chenghou Jin**[3], **Shaowei Cai**[4,5,*] and **Qingwei Lin**[6]
[1]School of Computer Science and Information Technology, Northeast Normal University, China
[2]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China
[3]Computer School, Beijing Information Science and Technology University, Beijing, China
[4]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[5]School of Computer Science and Technology, University of Chinese Academy of Sciences, China
[6]Microsoft Research, China
yiyuanwangjlu@126.com, jinchenghou1123@126.com, caisw@ios.ac.cn, qlin@microsoft.com

Abstract

The subgraph isomorphism problem (SIP) is a challenging problem with wide practical applications. In the last decade, despite being a theoretical hard problem, researchers designed various algorithms for solving SIP. In this work, we propose three main heuristics and develop an improved exact algorithm for SIP. First, we design a probing search procedure to try whether the search procedure can successfully obtain a solution at first sight. Second, we design a novel matching ordering as a value-ordering heuristic, which uses some useful information obtained from the probing search procedure to preferentially select some promising target vertices. Third, we discuss the characteristics of different propagation methods in the context of SIP and present an adaptive propagation method to make a good balance between these methods. Experimental results on a broad range of real-world benchmarks show that our proposed algorithm performs better than state-of-the-art algorithms for the SIP.

1 Introduction

The (non-induced) subgraph isomorphism problem (SIP), which is also known as the subgraph matching problem, involves deciding if there exists a copy of a pattern graph in a target graph. As one of the basic concepts of graph theory, the SIP can be seen as a generalization of both the maximum clique problem and the problem of testing whether a graph contains a Hamiltonian cycle. Recently, the SIP has been used in various domains, such as symbol recognition [Lladós *et al.*, 2001], social networks [Snijders *et al.*, 2006], computer vision [Damiand *et al.*, 2011], biochemical data [Bonnici *et al.*, 2013], RDF query processing [Kim *et al.*, 2015] and graph databases [Wang *et al.*, 2022]. For example, the SIP has also been used in the field of cheminformatics to search for some similarities between chemical compounds from their structural formula [Ohlrich *et al.*, 1993].

It is well known that the SIP is NP-complete in the general case [Johnson and Garey, 1979]. For the optimized version of the SIP, i.e., the maximum common induced subgraph (MCS) problem, many methods have been presented to deal with the MCS problem [McCreesh *et al.*, 2016a; McCreesh *et al.*, 2017; Liu *et al.*, 2020; Gocht *et al.*, 2020; Zhou *et al.*, 2022; Liu *et al.*, 2022]. The MCS approaches can be directly used into solving the SIP, but they usually have poor performance practically due to the characteristics of the SIP as the decision problem. Thus, up to now, the SIP has been still considered as a challenging problem.

In the last decade, lots of researchers focused on designing several exact methods to address the SIP [Zampelli *et al.*, 2010; Solnon, 2010; Bonnici *et al.*, 2013; Audemard *et al.*, 2014; McCreesh and Prosser, 2015; Carletti *et al.*, 2017; McCreesh *et al.*, 2018; Archibald *et al.*, 2019; Solnon, 2019; McCreesh *et al.*, 2020]. We list some representative solvers for the SIP as below. An early algorithm for the SIP named VF2 was proposed, which used a state space representation of the matching process and introduced a set of five feasibility rules for pruning the search tree [Cordella *et al.*, 2004]. Bonnici et al. [2013] developed a new search strategy called RI based on the pattern graph topology, which significantly reduced the search space without using any complex pruning rules or reduction procedures. Solnon [2010] introduced a new filtering algorithm called LAD based on local all-different constraints. The LAD algorithm was further improved by combining the local all-different constraints with the exploitation of path length properties, resulting in the PathLAD algorithm [Kotthoff *et al.*, 2016]. Very recently, Kraiczy and McCreesh [2021] improved the Glasgow [McCreesh *et al.*, 2020] by using a new form of filtering based upon clique-finding and designed a new algorithm called Glasgow+Clq. According to the literature, the current best algorithm for the SIP is Glasgow+Clq [Kraiczy and McCreesh, 2021].

*Corresponding author

1.1 Our Contribution

Motivated to contribute to further improving the performance of SIP, in this work, we choose the PathLAD algorithm as a baseline algorithm. Our proposed algorithm is divided into two parts, including a probing search procedure and a main search procedure. Below are three main novel ideas in our proposed algorithm.

First, we propose a probing search procedure in which the algorithm tries several times to judge whether the pattern graph is isomorphic to a subgraph of the target graph quickly. It has two main purposes. On the one hand, if this procedure can successfully solve a given instance, we can obtain an outcome within a short time. On the other hand, if this procedure cannot get any outcomes (i.e., reaching cutoff time), instead of using some traditional restart mechanisms, we can still extract the information from this search procedure to guide a main search procedure. That is, the algorithm learns about the useful search information between pattern vertices and target vertices and then uses this information in our proposed main procedure.

Second, we design a new matching ordering method to decide which target vertex from the domain of the corresponding pattern vertex is selected. Recently, several matching ordering methods were proposed [Archibald *et al.*, 2019; Wang *et al.*, 2022]. For example, Archibald et al. [2019] found that it is effective to preferentially select vertices with high degree values when selecting a matching target vertex. We follow this line of research by attempting to apply the degree information of target vertices in the matching process. At the same time, we also use the useful search information generated from the probing search procedure as another matching criterion. Thus, our proposed matching ordering method considers the above two principles, resulting in a novel scoring function denoted as $oscore$ used in the matching process.

Third, we present an adaptive propagation method to dynamically use different strong propagation methods for the SIP. Previous algorithms have always applied strong propagation methods to remove some unnecessary vertices from the corresponding domains, but these propagation methods need to cost lots of run time in practice, which reduces the performance of these algorithms. In some cases, instead of strong propagation methods, some weak propagation methods can make backtrack quickly or reduce the size of the corresponding domain effectively on some branches. Conversely, the performance of these algorithms would be also bad if they don't use any strong propagation methods because the algorithms fail to backtrack some branches immediately. Based on the above considerations, we analyze the properties of strong propagation methods and combine the search information generated from the main search procedure to dynamically employ different strong propagation methods during the search. To our best knowledge, it is the first time that different propagation methods are dynamically used to accelerate the search procedure for addressing SIP.

By incorporating these ideas, we develop an improved exact algorithm for the SIP called PathLAD+. Extensive experiments are carried out to evaluate PathLAD+ on the benchmarks used in the literature. Experimental results show that PathLAD+ outperforms four state-of-the-art SIP algorithms for all the benchmarks. In addition, our experimental analyses report that the proposed strategies play important roles in the outstanding performance of our proposed algorithm.

In the next section, we introduce some necessary background knowledge. After that, we present our proposed algorithm and its components. Experimental results are shown in Section 4. Finally, we make conclusions.

2 Preliminaries

2.1 Basic Definitions and Notations

Let $G = (V, E)$ be an undirected graph where a vertex set is $V = \{v_1, v_2, \ldots, v_n\}$ and an edge set $E = \{e_1, e_2, \ldots, e_m\}$. Each edge e is a 2-element subset of V, i.e., $e = (v, u)$. For an edge $e = (v, u)$, we say vertices v and u are the *endpoints* of edge e. For a vertex $v \in V$, the neighborhood of vertex v is denoted as $N_G(v) = \{u \mid (v, u) \in E\}$ and its degree is denoted as $deg_G(v) = |N_G(v)|$. A finite walk is a sequence of edges $(e'_1, e'_2, \ldots, e'_{q-1})$ for which there is a sequence of vertices $(v'_1, v'_2, \ldots, v'_q)$ such that $e'_i = (v'_i, v'_{i+1})$ for $i \in [1, q-1]$. A path is a finite walk in which all vertices and all edges are distinct, denoted as $\zeta^G = (v'_1, v'_2, \ldots, v'_q)$. The length of ζ^G is denoted as $|\zeta^G| = q$.

Given a pattern graph $G_p = (V_p, E_p)$ and a target graph $G_t = (V_t, E_t)$, the SIP is to decide whether G_p is isomorphic to some subgraph of G_t. Formally, the aim of the SIP is to obtain an injective matching f: $V_p \to V_t$ that associates a different target vertex to each pattern vertex, and preserves pattern edges, i.e., $(f(v), f(u)) \in E_t$ for $\forall (v, u) \in E_p$. It is noted that the subgraph is not necessarily induced, which means that two pattern vertices that are not linked by an edge may be matched to two target vertices that are linked by an edge. During the search procedure, the current matched list of pattern and target pairs is denoted as $D = \{\{v_1^p, v_1^t\}, \ldots, \{v_r^p, v_r^t\}\}$. For a pattern vertex $v^p \in V_p$, the *domain* of vertex v^p is defined as the set of target vertices that may be matched to v^p, i.e., $Dom(v^p) = \{v_1^t, v_2^t, \ldots, v_l^t\}$, and the size of its domain is $|Dom(v^p)| = l$.

2.2 Some Propagation Methods for the SIP

Recently, three filtering propositions [Zampelli *et al.*, 2010; McCreesh and Prosser, 2015] have been used in Glasgow+Clq [Kraiczy and McCreesh, 2021]. We first introduce three propositions that are used to judge whether the pattern vertices can be matched to the corresponding target vertices.

Proposition 1. *Given a pattern graph $G_p = (V_p, E_p)$ and a target graph $G_t = (V_t, E_t)$, if $v^p \in V_p$ can be matched to $v^t \in V_t$ (i.e., $f(v^p) = v^t$), it must satisfy $deg_{G_p}(v^p) \leq deg_{G_t}(v^t)$.*

Proposition 2. *Given a pattern graph $G_p = (V_p, E_p)$ and a target graph $G_t = (V_t, E_t)$, if $v^p \in V_p$ can be matched to $v^t \in V_t$ (i.e., $f(v^p) = v^t$), it must satisfy the i-th value of $ND(v^t)$ is not less than the same position of $ND(v^p)$ where $ND(v^p) = \{deg_{G_p}(u^p) \mid u^p \in N_{G_p}(v^p)\}$, $ND(v^t) = \{deg_{G_t}(u^t) \mid u^t \in N_{G_t}(v^t)\}$ and the positions of elements in $ND(v^p)$ and $ND(v^t)$ both are arranged in a descending order of the degree values.*

Proposition 3. *Given a pattern graph $G_p = (V_p, E_p)$ and a target graph $G_t = (V_t, E_t)$, if v^p and u^p in V_p can be matched to v^t and u^t in V_t (i.e., $f(v^p) = v^t$ and $f(u^p) = u^t$) as well as $Path^p$ and $Path^t$ are not empty, it must satisfy $|Path^p| \leq |Path^t|$ where $Path^p = \{\zeta^p \mid \zeta^p = (v^p, \ldots, u^p), |\zeta^p| = 3\}$ and $Path^t = \{\zeta^t \mid \zeta^t = (v^t, \ldots, u^t), |\zeta^t| = 3\}$.*

Some propagation methods of difference constraints [Solnon, 2010] are used in the PathLAD [Kotthoff *et al.*, 2016], which are shown as below.

- **Vertex constraint denoted as *FC(Diff)***: whenever a pattern vertex v^p is matched to a target vertex v^t, *FC(Diff)* removes v^t from the domains of all non-matched pattern vertices. The time complexity of *FC(Diff)* is $O(|V_p|)$.
- **Edge constraint denoted as *FC(Edges)***: whenever a pattern vertex v^p is matched to a target vertex v^t, *FC(Edges)* removes any target vertex not adjacent to v^t from the domain of every pattern vertex adjacent to v^p. The time complexity of *FC(Edges)* is $O(deg_{G_p}(v^p) \cdot |V_t|)$.
- **Global neighborhood constraint denoted as *GAC(allDiff)***: It ensures that all pattern vertices can be assigned to different target vertices. In detail, if a set of k pattern vertices can be found with only k target vertices among the domains of their pattern vertices, then those target vertices can be removed from the domains of other pattern vertices. The time complexity of *GAC(allDiff)* is $O(|V_p|^2 \cdot |V_t|^2)$.
- **Filtering method denoted as *LAD-filtering***: for $v^t \in Dom(v^p)$, a bipartite graph is defined as $G_{(v^p,v^t)} = (N_{G_p}(v^p), N_{G_t}(v^t), E_{(v^p,v^t)})$ where $E_{(v^p,v^t)} = \{(v', u') | v' \in N_{G_p}(v^p), u' \in N_{G_t}(v^t), u' \in Dom(v')\}$. If there does not exist a matching of the bipartite graph $G_{(v^p,v^t)}$ that covers $N_{G_p}(v^p)$, the pattern vertices adjacent to v^p cannot be matched to all different target vertices and thus v^t can be removed from $Dom(v^p)$. The time complexity of *LAD-filtering* is $O(|V_p| \cdot |V_t| \cdot deg^2_{G_p}(v^p) \cdot deg^2_{G_t}(v^t))$.

Note that two strong propagation methods *LAD-filtering* and *GAC(allDiff)* are implemented by the matching algorithm Hopcroft and Karp, and more details can be seen [Solnon, 2010].

3 The PathLAD+ Algorithm

This section describes the proposed PathLAD+ algorithm in Algorithm 1. Details of important functions in PathLAD+ will be presented in the following subsections. We use $switchL$ and $switchA$ to control whether the proposed algorithm uses *LAD-filtering* and *GAC(allDiff)*, respectively. Meanwhile, $nbnodes$ records the sum of call times and backtrack times of *SearchSIP*, $nbfail$ records the number of backtrack times of *SearchSIP*, and Nb is used in our proposed adaptive propagation method. The output value st has three values: *true* means that the algorithm can return a successful matched list; *false* means the pattern graph is not isomorphic to any subgraph of the target graph; *unknown* means that the algorithm cannot solve a given instance within a cutoff time. In the beginning, the algorithm reduces the domain of pattern vertices in a given pattern graph according to Propositions 1 and 2. If any domain becomes an empty set, the algorithm returns *false*. Otherwise, six variables are initialized accordingly (Lines 5–7). Then, the proposed algorithm can be divided into two procedures, including a probing search procedure (*ProSearch* in Line 8) and a main search procedure (*SearchSIP* in Line 9).

Algorithm 1 PathLAD+

Input: Pattern graph G_p, target graph G_t and the *cutoff* time
Output: outcome st

1: reduce the domain of pattern vertices based on **Propositions 1 and 2**;
2: **if** some domains become empty **then**
3: **return** *false*;
4: **end if**
5: $nbnodes := nbfail := 0$ and $Nb := +\infty$;
6: $switchL := switchA := 1$;
7: $st :=$ *unknown*;
8: *ProSearch*(G_p, G_t);
9: **return** *SearchSIP*$(G_p, G_t, \emptyset,$ *cutoff*$)$;

3.1 The Search Framework for SIP

The main function *SearchSIP* is shown in Algorithm 2, which is a recursive function. The input variable D is denoted as an already-matched list. If all vertices in the pattern graph are matched, which means that the algorithm has found a matched list for all pattern vertices, the algorithm returns *true* (Lines 1–2). Otherwise, if the time limit is reached, the algorithm returns *unknown* (Lines 3–4). The value of $nbnodes$ is increased by 1 (Line 6). The algorithm chooses a non-matched pattern vertex v_i^p with the smallest domain size, breaking ties by picking the one with the biggest degree value (Line 7). Afterward, the algorithm arranges the positions of target vertices in $Dom(v_i^p)$ based on a novel matching ordering method (i.e., $oscore$), which will be introduced in Section 3.3 (Line 8). In Lines 9–23, the algorithm tries to match each target vertex in $Dom(v_i^p)$ orderly. Before executing Line 9, the algorithm will store the domain of all pattern vertices. In Line 14, the algorithm restores the domain of all pattern vertices to their previous saved domain in Line 9. The algorithm orderly tries to match a target vertex v_i^t in the $Dom(v_i^p)$ to the selected pattern vertex v_i^p (Line 10). In each time, the algorithm reduces the domain of each non-matched pattern vertex based on an adaptive propagation method *APM*, which will be mentioned in Section 3.4 (Line 11). If the domain of some pattern vertex becomes empty, which means that v_i^t cannot be matched to v_i^p, the algorithm will restore and then continue to select the next target vertex in $Dom(v_i^p)$ (Lines 12–16). The corresponding values of $nbfail$ and $nbnodes$ will be increased by 1 (Line 13). If the algorithm doesn't obtain any empty domains, it will search for the next pattern vertex (Line 17). st stores the backtracking result of *SearchSIP*. If st equals *false*, the algorithm needs to restore the related domains and then continue to select the next target vertex (Lines

Algorithm 2 SearchSIP

Input: Pattern graph G_p, target graph G_t, an already-matched list of pattern and target pairs $D = \{\{v_1^p, v_1^t\}, \ldots, \{v_{i-1}^p, v_{i-1}^t\}\}$ and the *cutoff* time
Output: outcome st

1: **if** all the pattern vertices have been matched to respective target vertices **then**
2: **return** *true*;
3: **else if** elapsed time $>$ *cutoff* **then**
4: **return** *unknown*;
5: **end if**
6: $nbnodes$++;
7: select a vertex v_i^p with the smallest domain size $|Dom(v_i^p)|$ from all non-matched pattern vertices, breaking ties by picking the one with the biggest degree value; /∗ *recording info values in ProSearch, see Sec. 3.2* ∗/
8: sort the position of vertices in $Dom(v_i^p)$ based on the descending order of $oscore$ values; /∗ *see Sec. 3.3* ∗/
9: **for** each target vertex $v_i^t \in Dom(v_i^p)$ satisfying **Proposition 3 do**
10: match v_i^t to v_i^p;
11: $Dom(v_j^p) :=$ *APM*(G_t, v_j^p) for each non-matched pattern vertex v_j^p;
12: **if** some domains become empty **then**
13: $nbfail$++ and $nbnodes$++;
14: restore the domain of some pattern vertices;
15: **continue**;
16: **end if**
17: $st :=$ *SearchSIP*$(G_p, G_t, D \cup \{v_i^p, v_i^t\},$ *cutoff*$)$;
18: **if** $st ==$ *false* **then**
19: restore the domain of some pattern vertices;
20: **else**
21: **return** st;
22: **end if**
23: **end for**
24: **return** *false*;

18–19).

3.2 The Probing Search Procedure for SIP

Before calling a main search procedure, a probing search procedure *ProSearch* plans to use less time to try to successfully solve an instance. If *ProSearch* can successfully solve an instance, the algorithm can return a matched list of pattern and target pairs quickly or can determine that the pattern graph is not isomorphic to a subgraph of the target graph. Otherwise, the algorithm can also grasp some useful information, denoted as *info* in our algorithm, from this search procedure, i.e., recording which target vertices are included in the domain of the corresponding pattern vertex during this procedure. It means that the information obtained from *ProSearch* can reflect which vertex pair has more potential.

The specific way of updating *info* values is presented as follows. At first, the *info* value of each pair of pattern and target vertices is initialized to 0. In the search procedure, assuming that we select a pattern vertex v^p, we scan all target vertices in the domain of v^p (Line 7 in Algorithm 2). For each target vertex v_i^t in $Dom(v^p)$, the *info*$(\{v^p, v^t\})$ is increased by 1.

The proposed probing search procedure *ProSearch* works as follows. During the search procedure of *ProSearch*, the algorithm uses four propagation methods and the third proposition which have already been introduced in Section 2.2 to reduce the domain of pattern vertices. *ProSearch* has two search modes. In the first mode, to explore the vertices in the deep depth of the search tree, the algorithm runs *SearchSIP* with a cutoff time of 10 seconds without sorting the domains in any way (i.e., the default lexicographical order). Because some SIP instances can be found with only a small number of conflicts, the heavy commitment to early branching choices made by backtracking search can be extremely costly for these instances [Archibald *et al.*, 2019]. Based on the above consideration, in the second mode, the algorithm runs *SearchSIP* 20 times with a cutoff time of 1 second each time. To diversify early branch selections, the second mode sorts the position of target vertices in the corresponding domains randomly each time.

According to preliminary experiments, we found updating *info* during the main search procedure caused some vertex pairs with high *info* values and led to the poor performance. Thus, we restrict updating *info* only in *ProSearch*.

3.3 A Novel Matching Ordering Method

In the search procedure of *SearchSIP*, among non-matched pattern vertices, we select a pattern vertex with the smallest domain size. After choosing a pattern vertex, the next key step is how to select a target vertex from the domain of the selected pattern vertex. Whatever matching ordering method is used, the method will only affect the performance for some instances that have a successful matched list, whereas it has no influence on some instances where a given pattern graph is not isomorphic to any subgraph of the target graph because a complete search must be performed.

During the probing search procedure, we use *info* to collect useful information on the relationship between pattern and target vertices. After some pattern vertices have already matched to different target vertices, we assume that a target vertex v_i^t is often included in the domain of a pattern vertex $Dom(v^p)$, i.e., *info*$(\{v^p, v_i^t\})$ with a high value. We believe that v_i^t has more potential to match v^p compared to other target vertices because a matched pair $\{v^p, v_i^t\}$ would bring few conflicts. This means that some other pattern vertices are more likely to successfully find the corresponding target vertices in the following search when v_i^t matches v^p. At the same time, we consider the structure information of the target graph such as the degree value in our proposed matching ordering method.

As a result, we have the notion of a novel ordering score, which is formally defined as follows.

Definition 1. *For a pattern graph $G_p = (V_p, E_p)$ and a target graph $G_t = (V_t, E_t)$, the ordering score function, denoted as $oscore$ is a function on $v^p \in V_p$ and $v_t \in Dom(v^p)$ such that*

$$oscore(v^p, v^t) = \textit{info}(\{v^p, v^t\}) + deg_{G_t}(v^t)$$

In our proposed matching ordering method, when the algorithm chooses a pattern vertex v^p, the positions of target vertices in the domain of a selected pattern are arranged in a descending order of the $oscore$ values (Line 8 in Algorithm 2). The proposed matching ordering method depends on the search information of *ProSearch*. In the experimental section, we will show that this method has outstanding performance compared to several other sorting methods.

3.4 Adaptive Propagation Method

For the SIP, Glasgow [McCreesh *et al.*, 2020] and PathLAD [Kotthoff *et al.*, 2016] have outstanding performance, but they are completely different search strategies. Especially, Glasgow combines a weak propagation method with a fast restart mechanism. According to our preliminary experiments, Glasgow can make at least 10^4 recursive calls per second for some instances. On the contrary, PathLAD uses a strong propagation method at each stage of the search, and thus it sometimes makes less than one recursive call per second when dealing with some large target graphs. Based on our observations, no current algorithms for the SIP use different strengths of propagation methods at different stages of the search. Thus, our motivation is to design a method that can flexibly use some propagation methods in the search.

In the case of *ProSearch* procedure, the algorithm always uses *LAD-filtering* and *GAC(allDiff)*. Both of them have high time complexity. Therefore, in the main search procedure, we design an adaptive propagation method to guide the use of strong propagation methods *GAC(allDiff)* and *LAD-filtering*. The pseudo-code of *APM* is shown in Algorithm 3.

Let us consider *LAD-filtering* first. We define a target graph to be sparse if the median of its vertex degrees is less than deg_m. In our work, deg_m is set to 20. When a pattern vertex v^p is matched to a target vertex $v_i^t \in Dom(v^p)$, *LAD-filtering* ensures that every pattern vertex in $N_{G_p}(v^p)$ can match different target vertices in $N_{G_t}(v_i^t)$. Its execution time is based on the degree values of v^p and v_i^t. Because $deg_{G_t}(v_i^t)$ must be larger than or be equal to $deg_{G_p}(v^p)$, we just need to focus on the degree of target vertex v_i^t. If the target graph is sparse, the execution time of *LAD-filtering* is reasonable and we think that using it at every stage of the main search procedure is feasible.

In other cases, if the algorithm often backtracks due to lots of conflicts, the algorithm can actually turn to use some weak propagation methods including *FC(Diff)* and *FC(Edges)* instead of strong propagation methods. Although using weak propagation methods may result in searching deeper on the wrong branch compared to strong propagation methods, the algorithm can backtrack faster because the complexity of these weak methods is quite low. For such cases, calling *LAD-filtering* multiple times during the main search procedure will waste a lot of computation time. In our work, we analyze whether backtracking often occurs in the main search procedure by observing the values of $nbnodes$ and $nbfail$. Meanwhile, we use a parameter max_tries as the upper bound of $nbnodes$. In detail, on the one hand, If $nbnodes$ is smaller than max_tries, it may occur in the early stage of the search procedure. Because the backtracking for branch selection is costly, we want to explore more conflicts by using *LAD-filtering* on the wrong branches as early as possible. On the other hand, if $nbnodes$ is larger than max_tries, the algorithm has already explored some parts of the whole search space. For this case, we think the relationship between $nbfail$ and $nbnodes$ can provide some useful information for a given instance. If the number of failed vertices in the search procedure is large (i.e., $nbfail/nbnodes > \beta_1$ where β_1 is a parameter), it means that the algorithm has already backtracked a lot and thus the algorithm no longer uses *LAD-filtering* (Lines 3–4).

Algorithm 3 APM

Input: Target graph G_t and a non-matched pattern vertex v^p
Output: The reduced domain $D(v^p)$ of v^p

```
1: reduce D(v^p) based on FC(Diff) and FC(Edges);
2: if G_t is not a sparse graph then
3:     if nbnodes > max_tries && nbfail/nbnodes > β_1 then
4:         switchL := 0;
5:         if switchL == 0 at the first time then
6:             Nb := nbnodes;
7:         end if
8:     end if
9:     if switchL == 0 && nbnodes > 2Nb && nbfail/nbnodes > β_2 then
10:        switchA := 0;
11:    end if
12: end if
13: if switchL == 1 then
14:    reduce D(v^p) based on LAD-filtering;
15: end if
16: if switchA == 1 then
17:    reduce D(v^p) based on GAC(allDiff);
18: end if
19: return D(v^p);
```

In the following, we consider another strong propagation method *GAC(allDiff)*. Although this constraint has high time complexity in theory, it is actually faster than *LAD-filtering* in most cases. We will explain this reason as below. *GAC(allDiff)* constructs a bipartite graph between pattern vertices and target vertices. If a pattern vertex v^p is matched to a target vertex v_i^t, *GAC(allDiff)* will remove v_i^t from the domain of some other pattern vertices and ensures that all pattern vertices can still match different target vertices. Removing a selected target vertex from the generated bipartite graph only needs to find the next free target vertices for some pattern vertices by looking for an augmenting path [Derigs, 1981]. In fact, the size of a given target graph is usually larger than that of a corresponding pattern graph. Thus, when the sizes of the two graphs are quite different, *GAC(allDiff)* is likely to be run in linear time.

In the main search procedure, after disabling the *LAD-filtering*, the algorithm begins to consider whether to disable the *GAC(allDiff)*. When *LAD-filtering* is forbidden for the first time, we use variable Nb to record the current value of $nbnodes$ (Lines 5–6). *GAC(allDiff)* will continue to be used until $nbnodes$ has been increased twofold since

LAD-filtering is disabled at the first time, i.e., $nbnodes > 2Nb$. At this time, if the algorithm still backtracks frequently (i.e., $nbfail/nbnodes > \beta_2$ where β_2 is a parameter), we disable *GAC(allDiff)* (Lines 9–10). In the subsequent search procedure, the algorithm doesn't employ any strong propagation methods including *LAD-filtering* and *GAC(allDiff)*.

Remark that, in our work, the switch of propagation methods is one-way. The strength of weak propagation methods increases significantly with search depth, so there is no need to switch back to using strong propagation methods when the search depth reaches a certain point. Based on our preliminary experiments, we found that one-way switching was both straightforward and effective, whereas two-way switching exhibited poor performance on some instances. Recently, researchers have developed dynamic choice methods for several well-known problems, such as CSP [Stergiou, 2021]. One crucial step in algorithm design is to dynamically combine various methods for a particular problem. It's worth noting that our method is the first to use a dynamic choice approach to select propagation methods for the SIP.

Here, we will summarize the impact of the parameters β_1 and β_2 on the propagator choices as below. Parameters β_1 and β_2 are two thresholds that define whether a given instance is easy to backtrack due to numerous conflicts. When the conflict ratio is larger than these two parameters, we turn to using some simple propagation methods to make backtrack fast. Specifically, a higher value of β_1 indicates a greater tolerance for conflicts, allowing us to use all propagation methods. However, if the conflict ratio surpasses β_1, we discard the *LAD-filtering* method. On the other hand, a larger value of β_2 implies a higher tolerance for conflicts to solely rely on the strong propagation method *GAC(allDiff)*. When the conflict ratio exceeds β_2, we also abandon the *GAC(allDiff)*.

4 Experimental Evaluation

In this section, we carry out experiments to evaluate PathLAD+ on a broad range of various benchmarks, compared against the state-of-the-art algorithms for the SIP.

4.1 Benchmarks

For our experiments, we select all used instances from [Kraiczy and McCreesh, 2021; Liu *et al.*, 2022] which can also download from the website[1]. In total, we choose 15396 instances, which can be grouped into 8 benchmarks.

- **images-CVIU11 (6278 instances):** This benchmark includes 43 pattern graphs and 146 target graphs, which have been generated from segmented images [Damiand *et al.*, 2011]. In the benchmark, pattern graphs have between 22 and 151 vertices, whereas target graphs have between 1072 and 5972 vertices.
- **meshes-CVIU11 (3018 instances)**: It is composed of 6 pattern graphs and 503 target graphs, which have been generated from meshes modeling 3D object [Damiand *et al.*, 2011]. The number of vertices for pattern graphs is from 40 to 199, while the number of vertices is from 208 to 5873.
- **images-PR15 (24 instances)**: There are 24 pattern graphs that have between 4 and 170 vertices and 1 target graph that has 4838 vertices. All the graphs have been derived from segmented images [Solnon *et al.*, 2015].
- **scalefree (100 instances)**: Each instance contains a target graph whose vertices are between 200 and 1000 and a pattern graph whose vertices are 90% of the vertices of the corresponding target graph. All the instances in the benchmark have been randomly generated using a power law distribution of degrees [Solnon, 2010].
- **si (1170 instances)**: Each instance is composed of a target graph (between 200 and 1296 vertices) and a pattern graph (between 20% and 60% of the vertices of the corresponding target graph). This benchmark is from bounded valence graphs, modified bounded valence graphs, 4D meshes, and random generated graphs [Solnon, 2010].
- **phase-transition (200 instances)**: These random instances are chosen to be close to the satisfiable-unsatisfiable phase transition. Pattern graphs have 30 vertices, while target graphs have 150 vertices [McCreesh *et al.*, 2016b].
- **LV (1176 instances)**: The selected 49 graphs whose vertices are between 10 and 128 are considered as pattern and target graphs, and this benchmark has already been used as the tested benchmark [Liu *et al.*, 2020]. These graphs have different properties [Solnon, 2010], such as connected, biconnected, triconnected, etc.
- **LargerLV (3430 instances)**: From the above 49 LV graphs as the pattern graph and the other 70 graphs as the target graph whose vertices are between 138 and 6671. More details of the target graphs can be seen on the website[2].

4.2 Experiment Setup

We compare PathLAD+ with four state-of-the-art SIP algorithms, including Glasgow+Clq [Kraiczy and McCreesh, 2021], PathLAD [Kotthoff *et al.*, 2016], RI [Bonnici *et al.*, 2013] and VF2 [Cordella *et al.*, 2004]. The codes of these competitors are kindly provided by the authors. Our source code is publicly available at github[3]. Our proposed algorithm and four competitors are all implemented in C++ and compiled by g++ with '-O3' option. All the algorithms are run on Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz 512GB RAM under CentOS 7.9. The cutoff time is 3600 seconds for each instance. According to our preliminary experiments, parameters max_tries, β_1, and β_2 are set to 1000, 0.85, and 0.8, respectively.

For each algorithm, we report the number of instances for each benchmark ($\#inst$) and the number of successful solved instances ($\#solved$). The bold values in the tables indicate the best solution among all the algorithms.

[1] http://liris.cnrs.fr/csolnon/SIP.html

[2] https://github.com/ciaranm/cpaior2021-finding-subgraphs-with-side-constraints/tree/main/instances/largerGraphs

[3] https://github.com/yiyuanwang1988/PathLAD-Plus

Benchmark	#inst	PathLAD+ #solved	Glasgow+Clq #solved	PathLAD #solved	RI #solved	VF2 #solved
images-CVIU11	6278	**6278**	**6278**	**6278**	**6278**	**6278**
meshes-CVIU11	3018	**3008**	2987	2983	2695	2647
images-PR15	24	**24**	**24**	**24**	**24**	**24**
scalefree	100	**100**	**100**	**100**	82	21
si	1170	**1170**	**1170**	1109	1163	886
phase-transition	200	**134**	128	44	31	0
LV	1176	**1139**	1136	1130	1039	811
LargerLV	3430	**3344**	3318	3300	3154	2505
#total	15396	**15197**	15141	14968	14466	13172

Table 1: Experiment results on all the benchmarks.

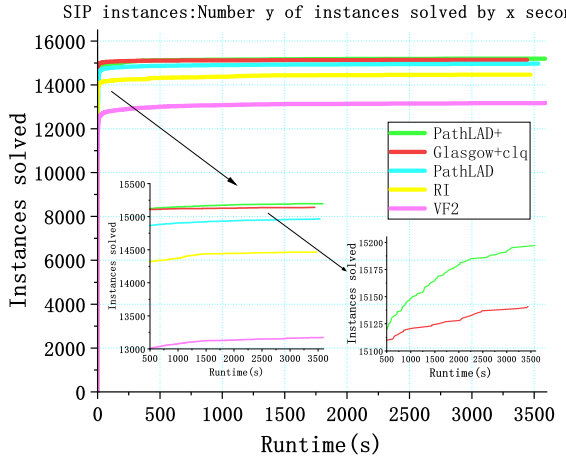

Figure 1: Detailed Results of PathLAD+ and all competitors on all the benchmarks.

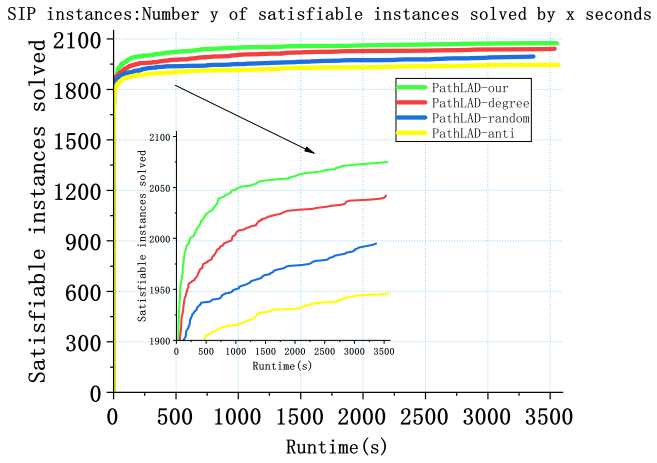

Figure 2: The run time of PathLAD with different matching ordering strategy on all the isomorphic satisfiable instances.

4.3 Experiment Results

We show the experiment results of our proposed algorithm and all competitors in Table 1. As observed from the results of Table 1, PathLAD+ performs much better than our baseline algorithm PathLAD on all the benchmarks. Overall, the performance of PathLAD+ totally dominates Glasgow+Clq, PathLAD, RI, and VF2. Because all algorithms can solve simple instances very well, we mainly focus on some hard instances. We can find that the performance of PathLAD+ is significantly better than all competitors on some hard benchmarks, especially in meshes-CVIU11. In this benchmark, all competitors have at least more than 30 unsolvable instances, whereas PathLAD+ only has 10 unsolvable instances within a cutoff time. Among the selected 15396 instances, PathLAD+ can solve 15197 instances within a cutoff time whereas the current best algorithm Glasgow+Clq can only solve 15141 instances. Furthermore, to intuitively display the performance of each algorithm, we report detailed results in Figure 1, which verifies the effectiveness of our proposed algorithm.

4.4 Analysis of Proposed Strategies

To confirm the effectiveness of our proposed matching ordering method, we evaluate different matching ordering methods on our baseline algorithm PathLAD, including 1) PathLAD-our uses our proposed matching ordering method; 2) PathLAD-degree selects a target vertex with the biggest degree value from the given domain; 3) PathLAD-random chooses a random target vertex from the given domain; 4) PathLAD-anti picks a target vertex with the smallest degree value from the given domain. Since different matching ordering methods only affect some isomorphic satisfiable instances [Archibald *et al.*, 2019], we have shown the performance of different matching ordering methods in these instances in Figure 2. Results show that our proposed matching ordering method performs better than other methods. Moreover, the proposed sorting method effectively utilizes the useful information generated from the probing search procedure, and it clearly improves the performance of SIP.

We compare PathLAD with one alternative algorithm PathLAD-1 that uses the adaptive propagation method. PathLAD-1 and PathLAD don't use any matching ordering methods, and the effectiveness of the adaptive propagation method can be clearly observed in Figure 3. The different

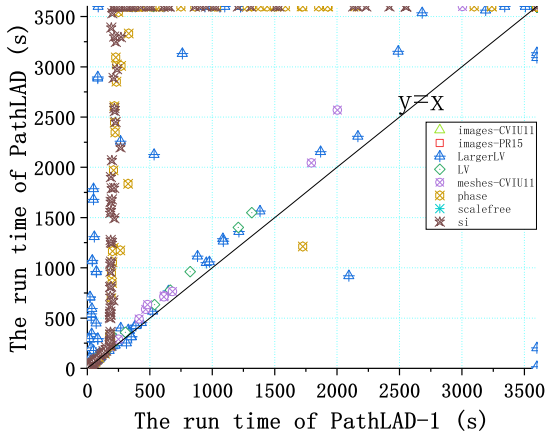

Figure 3: The run time of PathLAD and PathLAD-1 on all the benchmarks.

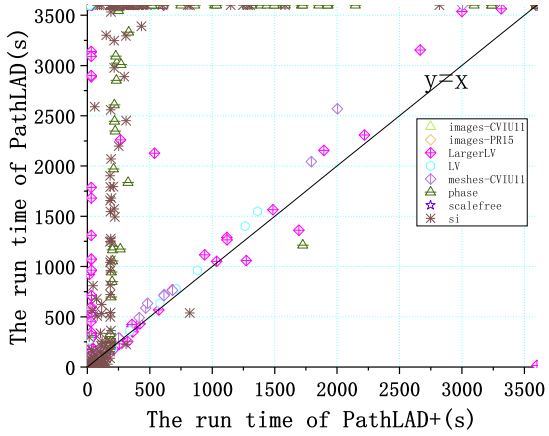

Figure 4: The run time of PathLAD+ and PathLAD on all the benchmarks.

colored points show the instance from the different benchmarks. Figures 2 and 3 intuitively show that the proposed two strategies make an important role in our proposed algorithm. Besides, because PathLAD is a baseline algorithm of our proposed algorithm, we compare PathLAD+ with PathLAD in terms of run time in Figure 4. Once again, the results show the superiority of PathLAD+.

5 Conclusion

In this paper, we propose a probing search procedure, a novel matching ordering method, and an adaptive propagation method for the SIP. Based on the above strategies, we develop an efficient algorithm called PathLAD+. Experiments show PathLAD+ significantly outperforms the state-of-the-art SIP algorithms.

As for future work, the proposed adaptive propagation method can be considered as a general idea to solve some other NP-hard problems [Chen *et al.*, 2023].

Acknowledgements

This work was supported by CAS Project for Young Scientists in Basic Research (Grant No.YSBR-040), NSFC (61806050), Jilin Science and Technology Association QT202005, and Science and Technology Development Program of Jilin Provice (YDZJ202201ZYTS412 and 20230101060JC). We would like to thank the anonymous referees for their helpful comments.

References

[Archibald *et al.*, 2019] Blair Archibald, Fraser Dunlop, Ruth Hoffmann, Ciaran McCreesh, Patrick Prosser, and James Trimble. Sequential and parallel solution-biased search for subgraph algorithms. In *CPAIOR*, pages 20–38, 2019.

[Audemard *et al.*, 2014] Gilles Audemard, Christophe Lecoutre, Mouny Samy-Modeliar, Gilles Goncalves, and Daniel Porumbel. Scoring-based neighborhood dominance for the subgraph isomorphism problem. In *CP*, pages 125–141, 2014.

[Bonnici *et al.*, 2013] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC bioinformatics*, 14(7):1–13, 2013.

[Carletti *et al.*, 2017] Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. Introducing vf3: A new algorithm for subgraph isomorphism. In *GbRPR*, pages 128–139, 2017.

[Chen *et al.*, 2023] Jiejiang Chen, Shaowei Cai, Yiyuan Wang, Wenhao Xu, Jia Ji, and Minghao Yin. Improved local search for the minimum weight dominating set problem in massive graphs by using a deep optimization mechanism. *Artificial Intelligence*, 314:103819, 2023.

[Cordella *et al.*, 2004] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

[Damiand *et al.*, 2011] Guillaume Damiand, Christine Solnon, Colin De la Higuera, Jean-Christophe Janodet, and Émilie Samuel. Polynomial algorithms for subisomorphism of nd open combinatorial maps. *Computer Vision and Image Understanding*, 115(7):996–1010, 2011.

[Derigs, 1981] Ulrich Derigs. A shortest augmenting path method for solving minimal perfect matching problems. *Networks*, 11(4):379–390, 1981.

[Gocht *et al.*, 2020] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *CP*, pages 338–357, 2020.

[Johnson and Garey, 1979] David S Johnson and Michael R Garey. *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, 1979.

[Kim *et al.*, 2015] Jinha Kim, Hyungyu Shin, Wook-Shin Han, Sungpack Hong, and Hassan Chafi. Taming subgraph isomorphism for rdf query processing. *Proceedings of the VLDB Endowment*, 8(11):1238–1249, 2015.

[Kotthoff *et al.*, 2016] Lars Kotthoff, Ciaran McCreesh, and Christine Solnon. Portfolios of subgraph isomorphism algorithms. In *LION*, pages 107–122, 2016.

[Kraiczy and McCreesh, 2021] Sonja Kraiczy and Ciaran McCreesh. Solving graph homomorphism and subgraph isomorphism problems faster through clique neighbourhood constraints. In *IJCAI*, pages 1396–1402, 2021.

[Liu *et al.*, 2020] Yanli Liu, Chu-Min Li, Hua Jiang, and Kun He. A learning based branch and bound for maximum common subgraph related problems. In *AAAI*, pages 2392–2399, 2020.

[Liu *et al.*, 2022] Yanli Liu, Jiming Zhao, Chu-Min Li, Hua Jiang, and Kun He. Hybrid learning with new value function for the maximum common subgraph problem. *arXiv preprint arXiv:2208.08620*, 2022.

[Lladós *et al.*, 2001] Josep Lladós, Enric Martí, and Juan J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1137–1143, 2001.

[McCreesh and Prosser, 2015] Ciaran McCreesh and Patrick Prosser. A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs. In *CP*, pages 295–312, 2015.

[McCreesh *et al.*, 2016a] Ciaran McCreesh, Samba Ndojh Ndiaye, Patrick Prosser, and Christine Solnon. Clique and constraint models for maximum common (connected) subgraph problems. In *CP*, pages 350–368, 2016.

[McCreesh *et al.*, 2016b] Ciaran McCreesh, Patrick Prosser, and James Trimble. Heuristics and really hard instances for subgraph isomorphism problems. In *IJCAI*, pages 631–638, 2016.

[McCreesh *et al.*, 2017] Ciaran McCreesh, Patrick Prosser, and James Trimble. A partitioning algorithm for maximum common subgraph problems. In *IJCAI*, pages 712–719, 2017.

[McCreesh *et al.*, 2018] Ciaran McCreesh, Patrick Prosser, Christine Solnon, and James Trimble. When subgraph isomorphism is really hard, and why this matters for graph databases. *Journal of Artificial Intelligence Research*, 61:723–759, 2018.

[McCreesh *et al.*, 2020] Ciaran McCreesh, Patrick Prosser, and James Trimble. The glasgow subgraph solver: Using constraint programming to tackle hard subgraph isomorphism problem variants. In *ICGT*, pages 316–324, 2020.

[Ohlrich *et al.*, 1993] Miles Ohlrich, Carl Ebeling, Eka Ginting, and Lisa Sather. Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. In *DAC*, pages 31–37, 1993.

[Snijders *et al.*, 2006] Tom AB Snijders, Philippa E Pattison, Garry L Robins, and Mark S Handcock. New specifications for exponential random graph models. *Sociological methodology*, 36(1):99–153, 2006.

[Solnon *et al.*, 2015] Christine Solnon, Guillaume Damiand, Colin De La Higuera, and Jean-Christophe Janodet. On the complexity of submap isomorphism and maximum common submap problems. *Pattern Recognition*, 48(2):302–316, 2015.

[Solnon, 2010] Christine Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artificial Intelligence*, 174(12-13):850–864, 2010.

[Solnon, 2019] Christine Solnon. Experimental evaluation of subgraph isomorphism solvers. In *GbRPR*, pages 1–13, 2019.

[Stergiou, 2021] Kostas Stergiou. Adaptive constraint propagation in constraint satisfaction: review and evaluation. *Artificial Intelligence Review*, 54(7):5055–5093, 2021.

[Wang *et al.*, 2022] Hanchen Wang, Ying Zhang, Lu Qin, Wei Wang, Wenjie Zhang, and Xuemin Lin. Reinforcement learning based query vertex ordering model for subgraph matching. In *ICDE*, pages 245–258, 2022.

[Zampelli *et al.*, 2010] Stéphane Zampelli, Yves Deville, and Christine Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15(3):327–353, 2010.

[Zhou *et al.*, 2022] Jianrong Zhou, Kun He, Jiongzhi Zheng, Chu-Min Li, and Yanli Liu. A strengthened branch and bound algorithm for the maximum common (connected) subgraph problem. In *IJCAI*, pages 1908–1914, 2022.