Efficient Subgraph Similarity All-Matching

Gaoping Zhu, Ke zhu, Wenjie Zhang, Xuemin Lin, and Chuan Xiao

The University of New South Wales, Sydney, NSW, 2052, Australia
{gzhu,kez,zhangw,lxue,chuanx}@cse.unsw.edu.au

Abstract. Being a fundamental problem in managing graph data, subgraph exact all-matching enumerates all isomorphic matches of a query graph q in a large data graph G. The existing techniques focus on pruning non-promising data graph vertices against q. However, the reduction and sharing of intermediate matches have not received adequate attention. These two issues become more critical on subgraph similarity all-matching due to the (possibly) massive number of intermediate matches. This paper studies the problem of efficient subgraph similarity all-matching by developing a novel query processing framework. We propose to effectively decompose a query graph into a hierarchical structure with the aim to minimize the number of intermediate matches and share intermediate matches. Novel techniques are then developed to estimate the number of intermediate matches, efficiently merge the intermediate matches, and generate efficient query execution plans. Experimental on real and synthetic datasets show that our approach outperforms the state-of-the-art approach for orders of magnitude.

1 Introduction

Graphs have been prevalently used in many applications for modeling complex data such as protein interaction networks (i.e., Bio-informatics), chemical compounds (i.e., Chem-informatics), social networks (i.e., Web), etc. Significant research efforts have been made towards many fundamental problems in managing graph data. The problem of *subgraph exact all-matching* is to enumerate all the exact matches (subgraph isomorphism mappings) of a query graph q in a large data graph G. This problem is of great importance in discovering graph structures and well studied in many previous works [16,18].

With the explosion of graph data, noisy or inconsistent data are unavoidable in many applications, while query graphs may also be noisy due to erroneous input. Consequently, subgraph exact all-matching may fail to find any exact matches and *subgraph similarity all-matching* is thus strongly demanded in such cases for approximate matches.

This paper studies efficient subgraph similarity all-matching; that is, to enumerate all *similarity matches* of a query graph q in a large data graph G by allowing at most δ missing edges (to be formally defined in Section 2). This problem stems from many applications. For example, in Bio-informatics, we can model protein interaction networks as graphs with proteins and interactions as vertices and edges, respectively. Given a noisy pathway query, our problem can

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 455–469, 2012.

return useful similarity matches in the network while no results can be found by subgraph exact all-matching. More applications can be found in [17].

Among all previous works on subgraph similarity matching [7,6,1,8,11,12,17], SAPPER [17] is the only one to enumerate all the similarity matches. It adopts the *enumerate-and-search* paradigm, which firstly identify all *feasible patterns* p (connected subgraphs of q missing at most δ edges of q) and then conducts subgraph exact all-matching to generate the final results. Although straight-forward to implement, the performance of the paradigm drops drastically when the number of intermediate matches is large, which is not uncommon in enumerating all matches. Hence, it is desirable to effectively (1) minimize the number of intermediate matches; and (2) share the intermediate matches to avoid redundant computation. We identify that (1) effective search order and (2) effective query decomposition are the keys to above two issues.

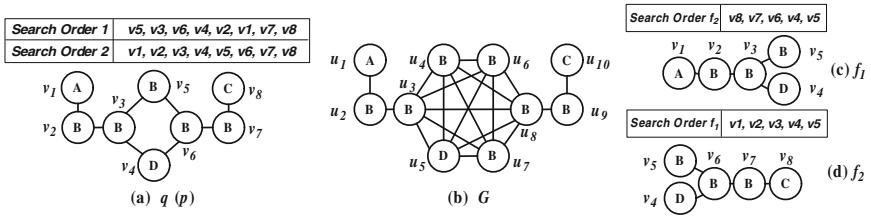


Fig. 1. Effective Search Order and Query Decomposition

Effective Search Order. Consider the query graph q in Figure 1(a) itself as a feasible pattern p, SAPPER conducts subgraph exact all-matching on p against the data graph G in Figure 1(b) by a *depth-first search* [3,9]. The two search orders in Figure 1(a) will encounter 350 and 47 intermediate matches, respectively. Hence, it is important to reduce the intermediate match number by using effective search order.

Effective Query Decomposition. Regarding the feasible pattern p in Figure 1(a) and its two decomposed fragments f_1 and f_2 in Figure 1(c)-(d). According to the search orders in Figure 1(c)-(d), enumerating the exact matches of f_1 and f_2 in G yields a total of 8 intermediate matches for f_1 (including 4 whole matches of f_1) and 8 for f_2 (including 4 whole matches of f_2). In merging the whole matches of f_1 with those of f_2, we only produce at most 16 more intermediate matches for p. Hence, the query decomposition further reduce the number to at most 32 intermediate matches. Moreover, we will show in Section 3 that we can also 'share' the computation cost of intermediate matches by query decomposition.

To the best of our knowledge, this is the first work to propose reducing the number of intermediate matches and sharing the computation of intermediate matches. The main contributions of this paper can be summarized as follows.

1. We propose a novel hierarchical framework DecQ to efficiently conduct subgraph similarity all-matching by decomposing the query into a set of unit sub-queries. We first compute the results of sub-queries (local matching) and then combine them to obtain the final results (global matching).

2. We propose a novel model to estimate the number of intermediate matches produced in local matching and then develop effective search order for sub-queries. In global matching, we develop an efficient merge-and-validation algorithm to combine the results of sub-queries by exploiting the computation sharing among overlapping feasible patterns.
3. We develop efficient heuristic algorithm to generate effective query decomposition for the sharing of intermediate matches.
4. We conducts extensive experiments on both real and synthetic datasets, which demonstrates that our approach outperforms the state-of-the-art approach by up to four orders of magnitude in terms of query response time.

Organizations. We organize the rest of this paper as follows. Section 2 presents important definitions and formalizes the problem. Section 3 proposes our hierarchical querying framework DecQ. Section 4 presents our local matching algorithm, while our global matching algorithm and effective query decomposition are studied in Section 5. Section 6 reports the experimental evaluation. Section 7 surveys related work and Section 8 concludes the paper.

2 Preliminaries

This paper studies *connected, vertex-labeled simple* graphs. A simple graph is an *undirected* graph with neither self-loops nor multiple edges. Without loss of generality, our approach can be easily extended to directed and/or edge-labeled graphs. Given a set Σ_V of labels, a graph g is defined as a triplet $(V(g), E(g), l)$ where $V(g)$ and $E(g) \subseteq V(g) \times V(g)$ are the set of vertices and undirected edges. If an edge is incident on $u, v \in V(g)$, $(u, v) \in E(g)$. The label function $l : V(g) \to \Sigma_V$ assigns a label $l(v)$ to each vertex $v \in V(g)$.

2.1 Problem Statement

Definition 1 (Subgraph Isomorphism Mapping). *Given two graphs $g = (V, E, l)$ and $g' = (V', E', l')$, a subgraph isomorphism mapping from g to g' is an injective function $f : V \to V'$ such that (1) $\forall v \in V$, $f(v) \in V', l(v) = l'(f(v))$; (2) $\forall (u, v) \in E$, $(f(u), f(v)) \in E'$.*

Given a subgraph isomorphism mapping from g to g', g is a *subgraph* of g' (g' is a *supergraph* of g), denoted by $g \subseteq g'$. We next define the graph edit distance.

Definition 2 (Graph Edit Distance). *Given two graphs g_1 and g_2, the* graph edit distance *$GED(g_1, g_2)$ from g_1 to g_2 is the minimum number of inserted edges required to transform g_1 to g_2.*

Note that: (1) The edit distance model is not symmetric. If g_1 cannot be transformed to g_2 by edge insertion, $GED(g_1, g_2) = +\infty$. (2) To control the number of similar graphs, we disallow label mismatch or vertex mismatch in the model. From now on, we abbreviate a query graph to a *query*. We next define the feasible pattern of q and similarity matches.

Definition 3 (Feasible Pattern Under δ). *Given a query q and a threshold δ, a feasible pattern of q under δ is a connected subgraph p of q such that $GED(p,q) \leq \delta$. The feasible pattern set $FP(q,\delta)$ consists of all the feasible patterns of q under δ.*

Definition 4 (Similarity Match). *Given a query q, a data graph G and a threshold δ, a similarity match of q in G is a subgraph isomorphism mapping from a feasible pattern p of q to G.*

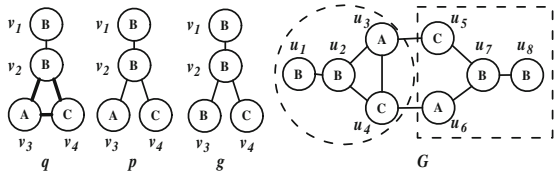


Fig. 2. Subgraph Similarity Matching

Example 1. Regarding Figure 2, assume $\delta = 1$. $p \in FP(q,1)$ because $GED(p,q) = 1$. $GED(q,p) = +\infty$ as we cannot transform q to p by edge insertion. $GED(g,q) = +\infty$ as the vertex v_3 of q can not be mapped into g. $FP(q,1)$ contains 4 feasible patterns (q and other 3 feasible patterns by deleting any bold edge in q). Both the two bounded matches in G are similarity matches of q. The circled one is also an exact match of q.

Problem Statement. Given a query q, a data graph G and a threshold δ, subgraph similarity all-matching returns a set S_q consisting of all the similarity matches of q in G.

Note that exact subgraph matching is a special case of subgraph similarity matching where $\delta = 0$. Let M_p denote the exact match set of each feasible pattern p. It is immediate that $S_q = \{ M_p | p \in FP(q,\delta) \}$.

3 A Hierarchical Framework

In this section, we propose a novel, three-phase framework DecQ for efficiently processing subgraph similarity all-matching. We summarize it as follows.

Phase 1: Query Decomposition. We decompose the query q into a hierarchical structure (Q,T) which implies a query execution plan. Here, Q is a set of connected, edge-disjoint subgraphs f of q called *fragments* and $\bigcup_{f \in Q} E(f) = E(q)$. Here, Q is also called an *edge-disjoint fragment* cover of q. T is a binary *decomposition tree* whose leaves correspond to all fragments in Q. Each internal node N in T represents a connected subgraph g of q, which can be further decomposed into two edge-disjoint subgraphs g_1 and g_2 residing on the two children. As to the query q in Figure 1, we can decompose q into $Q = \{ f_1, f_2 \}$ and obtain the decomposition tree as in Figure 3(a).

Phase 2: Local Matching. For each fragment $f \in Q$, we first compute its *local pattern set* $LP(f,\delta)$ consists of all local patterns f' (subgraphs of f missing at most δ edges of f). By using depth-first search, we compute the exact

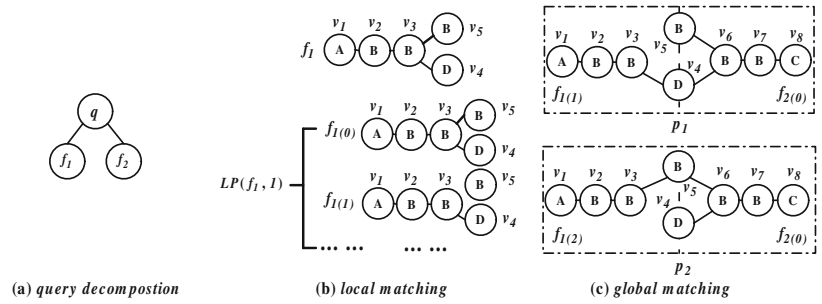


Fig. 3. Our Framework

matches $M_{f'}$, which are called *local matches* (similarity matches of f). For the completeness of final results, we allow f' to be disconnected and compute the exact matches of each component of f' in such case. In Figure 3(b), we have to compute the exact matches for all local patterns including $f_{1(0)}$ and $f_{1(1)}$.

Phase 3 : Global Matching. To distinguish the terms used in local matching, we call the feasible pattern p a *global pattern* and $FP(g, \delta)$ the *global pattern set*, respectively. Given $Q = \{ f_1, ..., f_m \}$, a global pattern p can be assembled from a set of local patterns $\{ f'_1, ...f'_m \}$ such that each $f'_i \in LP(f_i, \delta)$. In global matching, we merge these intermediate matches (local matches) $M_{f'_1}, ..., M_{f'_n}$ to obtain the exact match set M_p of p. Such exact matches of p are similarity matches of q and called *global matches*. Note that the query decomposition provides us an opportunity to share the computation cost of intermediate matches among various global patterns. After all global patterns have been processed, $S_q = \{ M_p | p \in FP(q, \delta) \}$ of q is returned.

For the query q in Figure 1(a), let $\delta = 1$. $p_1 = \{f_{1(1)}, f_{2(0)}\}$ and $p_2 = \{f_{1(2)}, f_{2(0)}\}$ in Figure 3(c)-(d) are two global patterns of q assembled from two local patterns. As p_1 and p_2 share $f_{2(0)}$, we only need to compute the local matches $M_{f_{2(0)}}$ once and share them in the global matching of p_1 and p_2.

4 Local Matching Algorithm

In this section, we propose a model to estimate the number of intermediate matches produced in local matching. We prove that problem of finding the optimal search order with minimized number of estimated intermediate matches is NP-hard and then develop effective search order to reduce the number of intermediate matches.

4.1 Estimating Intermediate Matches

Given a local pattern $f' \in LP(f, \delta)$ and the data graph G, assume a depth-first search algorithm A iteratively searches mappings for each $v \in V(f')$. The number of intermediate matches $|I_{f'}|$ produced in A varies greatly on different search orders employed by A. Although we can not obtain either $|I_{f'}|$ or $|M_{f'}|$ without applying A on f', we propose a novel model to estimate both of them.

For each vertex v (edge e) in a local pattern f', let $M(v)$ ($M(e)$) be the set of its vertex (edge) mappings in G. For each edge $(u, v) \in E(f')$, given any $u' \in M(u)$ and $v' \in M(v)$, the probability that there is an edge $(u', v') \in E(G)$ can be captured by Equation(1).

$$\theta(e) = \begin{cases} \frac{|M(e)|}{|M(v)| \times |M(u)|} & l(u) \neq l(v) \\ \frac{|M(e)|}{|M(v)| \times (|M(u)|-1)} & l(u) = l(v) \end{cases} \quad (1)$$

Given a search order on $V(f')$ according to which algorithm A searches vertex mappings, let $ig(f', k)$ be the subgraph of f' induced by the first k vertices in $V(f')$. We estimate $|M_{f'}|$ and $|I_{f'}|$ by Equation(2) and (3). Particularly, $|I_{f'}|$ is the summation of $|M_{ig(f',k)}|$ for each resulted induced subgraph $ig(f', k)$ along the search order.

$$\mathcal{E}(M_{f'}) = \prod_{v \in V(f')} |M(v)| \times \prod_{e \in E(f')} \theta(e) \quad (2)$$

$$\mathcal{E}(I_{f'}) = \sum_{i=1}^{|V(f')-1|} \mathcal{E}(M_{ig(f',k)}) \quad (3)$$

Example 2. Consider the fragment f_1 and data graph G in Figure 1(a)-(b), assume we only consider vertex label for matching vertices and edges. Figure 4 summarizes M_v and $\theta(e)$ for each $v \in V(f_1)$ and $e \in E(f_1)$. Let the search order on $V(f_1)$ be an ascending order on vertex ID. By Equation 2, $\mathcal{E}(M_{f_1}) = 7^3 \times 1 \times 1 \times (\frac{1}{7}) \times (\frac{5}{7}) \times (\frac{24}{42})^2 = 11.4$. By Equation 3, $\mathcal{E}(I_{f_1}) = 1+1+4+2.8 = 8.8$.

v	$\lvert M(v)\rvert$
v_1	1
v_4	1
other v	7

e	$\theta(e)$
(v_1, v_2)	$\frac{1}{1\times7} = 0.143$
(v_3, v_4)	$\frac{5}{1\times7} = 0.714$
other e	$\frac{24}{7\times6} = 0.571$

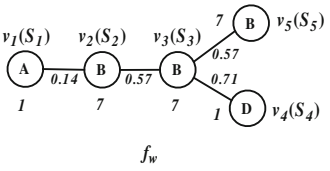


Fig. 4. Effective Search Order

Theorem 1. *Given any fragment f, the problem of finding the optimal search order with minimum estimated number of intermediate matches is NP-hard.*

Proof Sketch. It is immediate that Theorem 1 can be proved by reduction from maximum clique. Due to the interest of space, we omit the proof here.

4.2 Effective Search Order

Seeing the difficulty in finding the optimal search order, we proposes a heuristic algorithm to obtain an effective search order. Given a local pattern f', we first transform it into a weighted graph f_w such that (1) $\forall v \in V(f_w)$, $w(v) = |M(v)|$; (2) $\forall e \in E(f_w)$, $w(e) = \theta(e)$. For any subgraph g of f_w, $\mathcal{E}(M_g)$ equals the produce

of all vertex and edge weights on g. Our algorithm iteratively selects a vertex $v \in V(f_w)$ into the current search order V, which results in a new subgraph g of f' induced by V. In each iteration, we greedily select the vertex such that the resulted g has minimum estimation $\mathcal{E}(M_g)$. Consequently, the algorithm aims to to minimize $\mathcal{E}(M_g)$ for each resulted induced subgraph along the search order. Our algorithm GenOrder is outlined in Algorithm 1.

Algorithm 1: GenOrder (f_w)

Input : f_w: a weighted graph;
Output : V: an ordered set of vertices, initially an empty set;
1 Pick any $v' \in V(f_w)$ s.t. $\nexists\ v \in V(f_w) \wedge w(v) < w(v')$;
2 $V := V \bigcup \{ v' \}, V(f_w) := V(f_w) - \{ v' \}$;
3 **while** $V(f_w) \neq \emptyset$ **do**
4 Pick any $v \in V(f_w)$ such that $\mathcal{E}(M_g)$ is minimized for the subgraph g of f' induced by $V \bigcup \{ v \}$;
5 $V := V \bigcup \{ v' \}, V(f_w) := V(f_w) - \{ v' \}$;
6 **return** V;

Complexity Analysis. Clearly, GenOrder needs $O(|V(f_w)|)$ iterations and runs in $O(|V(f_w)||E(f_w)|)$. The space requirement is $O(|V(f_w)| + |E(f_w)|)$.

Example 3. Regarding the weighted graph f_w in Figure 4 transformed from f_1 in Figure 1(c), we weight all the vertices and edges following the left table. GenOrder will select S_i as the i-th vertex in the search order.

4.3 Efficient Local Matching

Enumerating Local Patterns. To conduct local matching, we must compute the local pattern set $LP(f, \delta)$ of each fragment $f \in Q$. Unlike the connected global patterns, any subgraph f' (connected or disconnected) of f missing no more than δ edges is a potential local pattern. This is because the connected components of f' may be bridges by other local patterns to form a global pattern. Given a *total order* on $E(f)$, for a subgraph f' of f missing at most δ edges, *key* of f' is defined as a set of 'ordered edges' missed in f; that is, $Key(f') = \{ e | e \in E(f) \wedge e \notin E(f') \}$. Below, we define a lexicographic order on $Key(f')$.

Definition 5 (Lexicographic Order). *Assume* $Key(f_a) = \{e_1^a, ..., e_k^a\}$ *and* $Key(f_b) = \{e_1^b, ..., e_l^b\}$ *represent two subgraphs* f_a, f_b *of a fragment* f. $Key(f_a)$ ¡ $Key(f_b)$ *if and only if, (1)* $Key(f_a) = \emptyset$*; or (2)* $\exists j \in [1, min(k, l)]$ *s.t.* $\forall i \in [0, j], e_i^a = e_i^b \wedge e_{j+1}^a < e_{j+1}^b$*; or (3)* $k < l$ *and* $\forall i \in [1, k], e_i^b = e_i^b$.

We enumerate all subgraphs f' in ascending order on $Key(f')$ and insert f' into $LP(f, \delta)$ if $Key(f')$ is not an edge cut of q. This is because f' can not be combined with other local patterns to for a connected global pattern. For disconnected local patterns f', we remove all isolated vertices from f' because these vertices must be presented in other fragments in Q.

Computing Local Matches. We extend the subgraph isomorphism test algorithm QuickSI [9] and adopt our effective search order to compute all local

matches. For local matches, we maintain $LP(f, \delta)$ in a *pattern table* $T(f)$ where each record $(Key(f'), M_{f'}) \in T(f)$ represents a local pattern f' and its exact match set. If f' is disconnected into a set $\{ c_i | 1 \leq i \leq n \}$ of n connected components, $M_{f'}$ is replaced with the exact match sets M_{c_i} of each c_i.

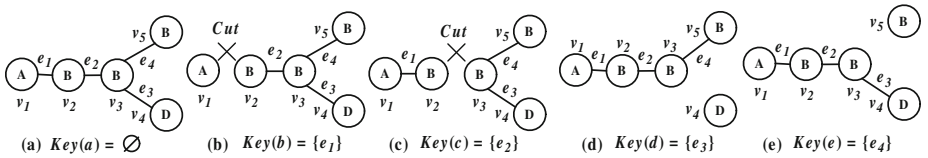


Fig. 5. Enumerating Local Patterns

Example 4. Regarding the fragment f_1 in Figure 1, assume $\delta = 1$. Figure 5$(a)-(e)$ show all 5 subgraphs of f_1 missing at most one edge. Since $Key(b)$ and $Key(c)$ are two edge cuts, $LP(f_1, \delta) = \{ (a), (d), (e) \}$. Finally, we remove v_4 and v_5 from (d) and (e) as they are single-vertex components.

5 Global Matching Algorithm

In this section, we propose an efficient merge-and-validation global matching algorithm, which shares the computation cost of intermediate matches of various global patterns. Then we develop effective query decomposition technique to maximize the computation sharing.

5.1 Enumerating Global Patterns

Similar as in Section 4, we assign any subgraph q' of q missing at most δ edges with a key $Key(q') = \{ e | e \notin E(q') \wedge e \in E(q) \}$ representing the ordered missing edges. The lexicographic order on $Key(q')$ can be similar defined. We organize all global patterns of q in a *pattern lattice* with with $|E(q)| + 1$ levels. Level-i contains the keys $Key(q')$ of all q' missing i edges in q. On the top and bottom level, we put $Key(q) = \emptyset$ and $E(q)$, respectively. Due to the error threshold δ, we can safely discard all the levels below the δ-th level. For any two subgraphs q_a and q_b from the i-th and $i+1$-th level, if q_a is obtained by removing an edge from q_b, we call q_a a child of q_b (q_b a parent of q_a). We order all the subgraphs q' on level-i in ascending lexicographic order on $Key(q')$. Clearly, if $Key(q')$ is an edge cut of q, q' is disconnected and thus not a global pattern; otherwise, q' is a global pattern which must fall in one of the following two categories.

- If $Key(q'')$ is an edge cut of q for all the children q'' of q', or q' has no children, we call such q' a **minimal pattern**.
- If $Key(q'')$ is not an edge cut of q for some child q'' of q', we call such q' a **non-minimal pattern**.

Example 5. Regarding the query q in Figure 6, we depict its pattern lattice for $\delta = 2$ with all the subgraphs represented by their keys. We bound all subgraphs q' with rectangles if $Key(q')$ is an edge cut of q. All the minimal patterns are circled, while all the global patterns above level-2 are non-minimal patterns.

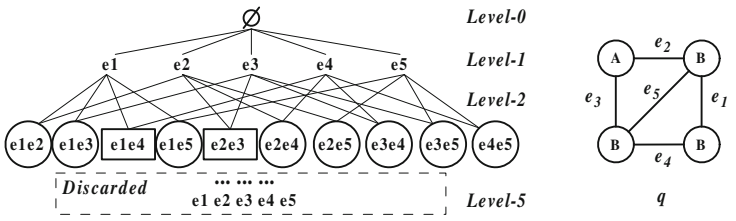


Fig. 6. Pattern Lattice

Our merge-and-validation matching algorithm is presented in Algorithm 2 which traverse the pattern lattice level by level, according to the order of $Key(q')$. The algorithm processes minimal patterns with sharing-aware merge, while efficient edge validation is adopted to process non-minimal patterns. We give details on the sharing-aware merge and edge validation in the following two subsections.

Algorithm 2: GlobalMatch (q, G, L, δ)

```
Input  : q: a query; G: a data graph; L: the pattern lattice; δ: the threshold;
Output : S_q: similarity match set of q;
for i := δ to 0 do
    for each q' in ascending order on level-i do
        if Key(q') is an edge cut then continue;
        if q' is a minimal pattern then
            Compute M_q' by sharing-aware merge;
        else
            Compute M_q' by edge validation;
        S_q := S_q ⋃ { M_q' };
return S_q;
```

5.2 Matching Minimal Patterns

We compute the exact matches M_p of minimal patterns p by merging the intermediate matches under according to the decomposition tree T in the query execution plan. Note that any internal node N in T indicates a connected subgraph g of q, which are further decomposed into two edge-disjoint subgraphs residing on the two child nodes L and R of N. Let $N.g$ be the subgraph represented by N. Let $J = V(L.g) \bigcap V(R.g)$ be the common vertices of $L.g$ and $R.g$. We compute $M_{N.g}$ by equi-joining $M_{L.g}$ and $M_{R.g}$ on J. In practice, we adopt hash join to perform the task. Generally, following (Q, T), we first decompose p along T in a top-down fashion to retrieve the decomposed local patterns corresponding to fragments in Q and then recursively merge the intermediate matches (local matches) to compute M_p.

Sharing Intermediate Matches. Given two minimal patterns p and p', if they share common intermediate patterns, we can share the merge cost of them. Such intermediate patterns are either local patterns of the merge results of a set of local patterns. We create an intermediate pattern table $T(N)$ on each

internal node N. For each newly encountered intermediate pattern $N.g'$ on N merged from some local patterns, we insert a tuple $(N.g', M_{N.g'})$ into $T(N)$. Consequently, if two patterns p and p' share $N.g'$, we can share the pre-computed $M_{N.g'}$ to avoid redundant merge cost.

Maintain Disconnected Intermediate Matches. Due to the possibly disconnected local patterns, an intermediate pattern $N.g'$ may not be connected. In such case, we maintain the exact matches for each of its component and delay their merge until a 'bridge' intermediate pattern links them at a later stage.

5.3 Matching Non-minimal Patterns

For a non-minimal patterns p at level-i, any child pattern p' of p at level-$i+1$ only miss one edge in p. According to Definition 2, any exact match of p must be an exact match of p'. Thus, we only need to conduct edge validation on $M_{p'}$ for computing M_p. We pick the child p' of p with minimum $|M_{p'}|$ and check each exact match $\mathcal{F} \in M_{p'}$ to see if the extra edge in p exists in $\mathcal{F}$. If so, $\mathcal{F}$ is also an exact match of M_p.

5.4 Effective Query Decomposition

Given a global pattern p decomposed into a set of local patterns, the computation cost of M_p contains (1) the search cost of local patterns and (2) the merge cost of intermediate patterns. The search cost of a local pattern f' can be evaluated by $\mathcal{E}(M_{f'}) + \mathcal{E}(I_{f'})$, while the merge cost of an intermediate pattern g' can also be evaluated by $\mathcal{E}(M_{g'})$. Equation(3) and (2) can be used to calculate the cost.

However, it is expensive to generate an optimal decomposition for each $p \in FP(q, \delta)$ is expensive as there are too many p and possible decompositions to consider. Hence, we propose *recursive bisection* to generate a uniform decomposition for all global patterns by considering q only.

Recursive Bisection. The recursive bisection works as follows. We initialize an empty query cover Q and a decomposition tree T with only one root node R representing q. The we recursively bisect q and its successive decomposed subgraphs to construct Q and T. For each newly decomposed subgraph g, we build a new leaf node N in T to hold g as a fragment. Consequently, the computation cost on N is simply the search cost of all local patterns g' of g. We the attempt to bisect g into g_1 and g_2, two smaller fragments to reduce the computation cost. The new computation cost contains (1) the search cost of local patterns of g_1 and g_2 and (2) the merge cost of the local patterns of g_1 and g_2.

Equation (4) and (5) estimate computation cost on N before and after the bisection, respectively. Note that we approximate the search cost of all local patterns by the search cost of their corresponding fragments. According to Definition 2, the number of possible local patterns of g is $\alpha_g = \Sigma_{i=0}^{\delta} \binom{|E(g)|}{i}$. Recall that we use $\mathcal{E}(M_g)$ to estimate the merge cost. Similarly, if we decompose g, there are at most α_g intermediate patterns to be merged. Equation 6 gives the cost gain of the bisection.

$$C_a = \alpha_g(\mathcal{E}(I_g) + \mathcal{E}(M_g)) \quad (4)$$

$$C_b = \alpha_{g_1}(\mathcal{E}(M_{g_1}) + \mathcal{E}(I_{g_1})) + \alpha_{g_2}(\mathcal{E}(M_{g_2}) + \mathcal{E}(I_{g_2})) + \alpha(g)(\mathcal{E}(M_g)) \quad (5)$$

$$C_g = \alpha_g\mathcal{E}(I_g) - \alpha_{g_1}(\mathcal{E}(M_{g_1}) + \mathcal{E}(I_{g_1})) - \alpha_{g_2}(\mathcal{E}(M_{g_2}) + \mathcal{E}(I_{g_2})) \quad (6)$$

Since $\mathcal{E}(I_g)$ is fixed with a pre-given search order before bisection, we aim to reduce the search cost of g_1 and g_2. Note that a good bisection should be balanced since both $\mathcal{E}(M_{g_1})$ and $\mathcal{E}(I_{g_2})$ has an exponential growth with the increase of graph size. Hence, our bisection always aims to bisect g into two connected subgraphs with approximately the same size. In experiments, we only bisect g when it yields a positive cost gain. The minimum fragment size is $\delta + 1$.

6 Performance Evaluation

In this section, we report our experimental results and analyses. We obtain the binary code of SAPPER from its authors [17]. All our algorithms are implemented in C++ and compiled with GCC 4.3.2 with `-O3` flag. All experiments are conducted on a PC with Intel Xeon 2.40GHz CPU and 4GB memory running Debian 4.1.1-21.

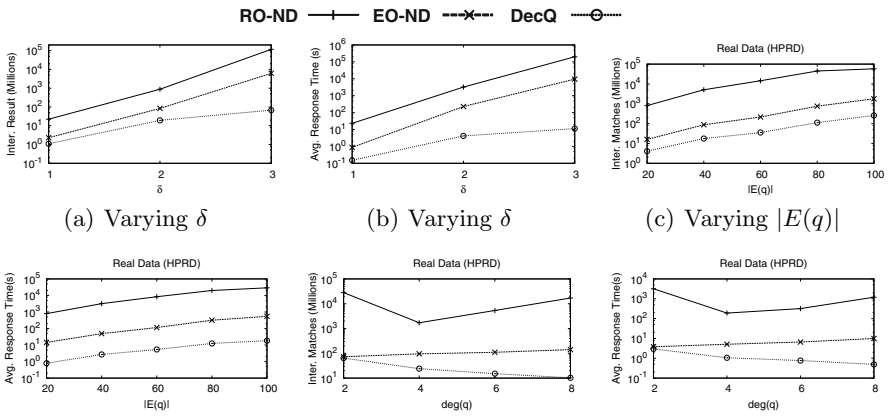
Datasets. Our real dataset is the Human Protein Interaction Network, a popular benchmark (`http://www.hprd.org/download`) for evaluating subgraph matching and search techniques. The network, denoted G_H, consists of $9,460$ vertices, $37,081$ edges and 307 distinct vertex labels. We adopt G_H to study the efficiency of our proposed algorithms. We generate synthetic data graphs and queries to study the scalability of our proposed algorithms varying data graph settings. Note that the queries are always generated by selecting induced data graphs from the underlying data graphs and we randomly insert $1-3$ 'noisy edges'. All query set contains 100 queries. The synthetic queries are similarly generated as the real queries. We summarize the default parameters of query and data graphs in Table 1. Note that $|V(G)|$, $deg(G)$, and $|\Sigma_V|$ are applicable for synthetic datasets only. The default error threshold δ is 2 unless otherwise specified.

Table 1. Default Values of Parameters

Parameters	$\|E(q)\|$	avg. $deg(q)$	$\|V(G)\|$	avg. $deg(G)$	$\|\Sigma V\|$
Default Values	40	4	5,000	12	100

Evaluated Algorithms. We evaluate the following algorithms in this paper: (1) RO-ND The basic subgraph similarity all-matching algorithm which enumerates feasible patterns of q and searches the exact matches of feasible patterns with random search order; (2) EO-ND The modified RO-ND algorithm equipped with effective search order. (3) DecQ Our proposed algorithm with effective search order and effective query decomposition. (4) SAPPER The algorithm developed in [17]. In order to facilitate the local matching, the indexing technique in [17] is applied on all algorithms to efficiently identify candidate data graph vertices.

6.1 Varying Error Threshold and Query Settings



(a) Varying δ (b) Varying δ (c) Varying $|E(q)|$

(d) Varying $|E(q)|$ (e) Varying $deg(q)$ (f) Varying $deg(q)$

Fig. 7. Varying Error Threshold and Query Settings

Varying Error Threshold δ. We compare 3 algorithms RO-ND, EO-ND, DecQ on G_H to study the effect of our effective search order and query decomposition. We plot the averaged number of intermediate matches and query response time in Figure 7(a) and 7(b). Clearly, the intermediate matches of RO-ND and EO-ND grow the fastest, while DecQ have decent growths. Thanks to the effective search order, EO-ND produces only up to 1/19 as many intermediate matches as RO-ND does. DecQ outperforms the other algorithms on all δ settings.

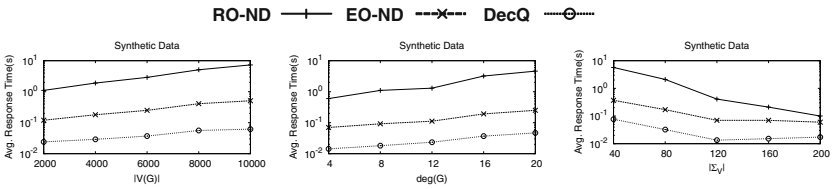
The trend of query response time confirms the effectiveness and efficiency of our proposed techniques. All algorithms costs more time for larger δ, while DecQ is the most efficient among the four. When $\delta = 3$, EO-ND is 21 times faster than RO-ND, while DecQ has an additional speed-up over EO-ND for up to 840 times. The gaps between DecQ and other algorithms increase when δ increases because more computation on overlapping global patterns can be shared.

Varying Query Size $|E(q)|$. We next evaluate the effect of query size on G_H. The intermediate matches and the query response time are plotted in Figure 7(c) and 7(d). The intermediate matches and the response time both increases with the query size. The reason is that we have to go deeper in the depth-first search for RO-ND and EO-ND, or decompose q into more fragments for DecQ; yet both our search order and query decomposition are effective over all $|E(q)|$ settings.

Varying Average Query Density $deg(q)$. We then evaluate the effect of query density on G_H and report the results in Figures 7(e) and 7(f). It is interesting that all algorithms exhibit different trends. The response time of RO-ND first decreases and then rebounds, while that of EO-ND almost levels over all density settings. The response time of DecQ keeps decreases when q becomes denser. Same trend is observed on the number of intermediate matches. There are two counteracting factors that affect this result: (1) the number of global patterns increases for denser queries; (2) dense global patterns are less likely to have

matches due to the strict topological structure. For RO-ND, the second factor is dominant for small degrees, while the first factor is more significant when $deg(q) > 4$. For EO-ND, the effective search order makes it more efficient to find matches and hence weakened the effect of the first factor. Thanks to the shared intermediate matches, the matching is even faster for DecQ, and hence the second factor dominates.

6.2 Varying Data Graph Settings



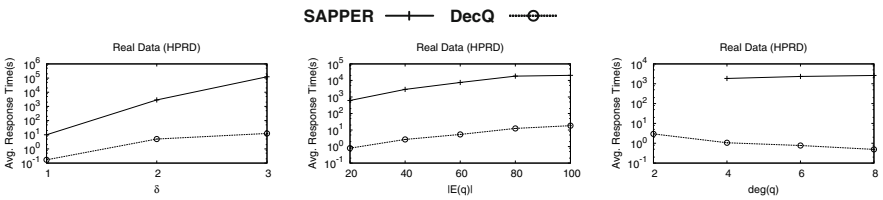
(a) Varying $|E(g)|$ (b) Varying $deg(g)$ (c) Varying $|\Sigma_V|$

Fig. 8. Varying Data Graph Settings

Varying Data Graph Size and Density. We firstly evaluate the effect of data graph size and density on synthetic dataset. The results with varying data graph size and density are reported in Figure 8(a) and 8(b). DecQ achieves 8 times speed-up against EO-ND and 118 times against RO-ND over all graph size settings. DecQ and EO-ND exhibit lower growth rate because the effective search order starts with the most selective vertex, whose number in G does not grow as fast as $|V(G)|$. Similar trend is observed on all density settings which confirms DecQ has better scalability than the other algorithms.

Varying Number of Vertex Labels. We report our results over different $|\Sigma_V|$ settings in Figure 8(c). All algorithms consumes less time when $|\Sigma_V|$ increases. This is because the vertices are rendered more selective and thus leads to few intermediate matches. The response time almost levels for both EO-ND and DecQ when the $|\Sigma_V|$ exceed 120. This is because the selectivity of the most label selective vertices in the search order barely changes when we include more labels.

6.3 Comparison with SAPPER



(a) Response Time (b) Response Time (c) Response Time

Fig. 9. Comparison with SAPPER

We finally compare DecQ with SAPPER by varying δ, $|E(q)|$ and $deg(q)$ on G_H. The results are reported in Figure 9(a) to 9(c). When $\delta = 3$, DecQ is faster than SAPPER by 4 orders of magnitude. We do not report the results of SAPPER on $deg(q) = 2$ since it runs out of all $4GB$ memory in storing intermediate matches. The large gap on response time between two algorithms is witnessed on all experiments. This is mainly due to our effective search order and query decomposition. The reduction and sharing of intermediate matches significantly save the cost for processing highly overlapping global patterns. Note that two algorithm exhibits different trend on query density settings. The main reason is that more global patterns are enumerated for denser queries, while they are more selective and less likely to have matches. Since DecQ shares the computation cost among global patterns, it is less sensitive to the effect of increasing global patterns. Consider both factors, the decreased response time can be explained.

7 Related Work

Many fundamental problems in managing graph data has been extensively studied. These include subgraph exact and similarity all-matching, subgraph/supergraph containment search and subgraph similarity search. On *exact subgraph all-matching*, most studies propose to build efficient index to prune non-promising data graph vertices against the query. [16] develops an indexing technique called GADDI to index nearby discriminative subgraphs as signatures, while shortest path are also adopted in [18] as unit index structure. To handle noisy graph data, [11] studies subgraph similarity all-matching by developing neighborhood-based index structure. [17] on the other hand, transform the problem to subgraph exact all-matching by proposing the enumerate-and-search paradigm.

Subgraph containment search [4,5,9,10,13,14,19,20] and supergraph containment search. [2,15] also attract great research interests. On *subgraph containment search*, [10] proposes the first filtering-verification framework by indexing path-features to filter false results before the expensive verification. [13] improves the filtering power by indexing discriminative graph-features. To further reduce filtering cost and index construction size, [19] and [14] independently propose to adopt tree-features. [9] proposes efficient verification approach to accelerate query processing. On *supergraph containment search*, [2] propose cIndex to select contrast features via query log, while [15] enhances the verification phase by sharing search cost on common subgraphs of data graphs. On *subgraph similarity search*, [12] follows the filtering-verification framework to remove false results by counting the number of missing features. Most recently, [8] proposes efficient verification algorithm and a novel filtering-validation-verification paradigm to process the problem.

8 Conclusions

In this paper, we study the problem of efficient subgraph similarity all-matching. We develop a hierarchical framework DecQ to firstly decompose the query into a set of unit sub-queries and then combine the results of sub-queries for final

results. We propose novel intermediate match estimation model and develop heuristic algorithm to generate effective search order for the reduction of intermediate matches. We develop a merge-and-validation algorithm to combine sub-query results by sharing the computation cost of intermediate matches. Our experimental results demonstrate that our proposed approach outperforms the state-of-the-art approaches by up to 4 orders of magnitude in terms of both intermediate match number and query response time.

References

1. Bunke, H., Foggia, P., Guidobaldi, C., Sansone, C., Vento, M.: A comparison of algorithms for maximum common subgraph on randomly connected graphs. In: SSPR/SPR, pp. 123–132 (2002)
2. Chen, C., Yan, X., Yu, P.S., Han, J., Zhang, D.-Q., Gu, X.: Towards graph containment search and indexing. In: VLDB, pp. 926–937 (2007)
3. Cordella, L., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In: 3rd Workshop on Graph-based Representations in Pattern Recognition, pp. 149–159 (2001)
4. He, H., Singh, A.K.: Closure-tree: An index structure for graph queries. In: ICDE, p. 38 (2006)
5. Jiang, H., Wang, H., Yu, P.S., Zhou, S.: Gstring: A novel approach for efficient search in graph databases. In: ICDE, pp. 566–575 (2007)
6. Krissinel, E.B., Henrick, K.: Common subgraph isomorphism detection by backtracking search. Softw. Pract. Exper. 34(6), 591–607 (2004)
7. McGregor, J.J.: Backtrack search algorithms and the maximal common subgraph problem. Softw. Pract. Exper. 12(1), 23–34 (1982)
8. Shang, H., Lin, X., Zhang, Y., Yu, J.X., Wang, W.: Connected substructure similarity search. In: SIGMOD, pp. 903–914 (2010)
9. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. PVLDB 1(1), 364–375 (2008)
10. Shasha, D., Wang, J.T.-L., Giugno, R.: Algorithmics and applications of tree and graph searching. In: PODS, pp. 39–52 (2002)
11. Tian, Y., Patel, J.M.: Tale: A tool for approximate large graph matching. In: ICDE, pp. 963–972 (2008)
12. Yan, X., Han, P.S.Y.J.: Substructure similarity search in graph databases. In: SIGMOD, pp. 766–777 (2005)
13. Yan, X., Yu, P.S., Han, J.: Graph indexing: A frequent structure-based approach. In: SIGMOD Conference, pp. 335–346 (2004)
14. Zhang, S., Hu, M., Yang, J.: Treepi: A novel graph indexing method. In: ICDE, pp. 966–975 (2007)
15. Zhang, S., Li, J., Gao, H., Zou, Z.: A novel approach for efficient supergraph query processing on graph databases. In: EDBT, pp. 204–215 (2009)
16. Zhang, S., Li, S., Yang, J.: Gaddi: distance index based subgraph matching in biological networks. In: EDBT, pp. 192–203 (2009)
17. Zhang, S., Yang, J., Jin, W.: Sapper: Subgraph indexing and approximate matching in large graphs. In: VLDB (2010)
18. Zhao, P., Han, J.: On graph query optimization in large networks. PVLDB 3(1), 340–351 (2010)
19. Zhao, P., Yu, J.X., Yu, P.S.: Graph Indexing: Tree + Delta >= Graph. In: VLDB, pp. 938–949 (2007)
20. Zou, L., Chen, L., Yu, J.X., Lu, Y.: A novel spectral coding in a large graph database. In: EDBT, pp. 181–192 (2008)