Efficient Algorithms for Node Disjoint Subgraph Homeomorphism Determination*

Yanghua Xiao, Wentao Wu, Wei Wang, and Zhenying He

Department of Computing and Information Technology
FuDan University, ShangHai, China
{Shawyanghua,wentaowu1984}@gmail.com,{weiwang1,zhenying}@fudan.edu.cn

Abstract. Recently, great efforts have been dedicated to researches on the management of large-scale graph-based data, where node disjoint subgraph homeomorphism relation between graphs has been shown to be more suitable than (sub)graph isomorphism in many cases, especially in those cases where node skipping and node mismatching are desired. However, no efficient algorithm for node disjoint subgraph homeomorphism determination (ndSHD) has been available. In this paper, we propose two computationally efficient ndSHD algorithms based on state spaces searching with backtracking, which employ many heuristics to prune the search spaces. Experimental results on synthetic data sets show that the proposed algorithms are efficient, require relatively little time in most of cases, can scale to large or dense graphs, and can accommodate to more complex fuzzy matching cases.

1 Introduction

Graph-based pattern matching is one of the key issues underlying large-scale graph-based data management, which recently has attracted more and more research interests, due to the broad applications of graph-based data. Existing graph pattern matchings based upon *subgraph isomorphism* cannot represent the fuzzy matching in some cases where *node skipping or node mismatching is allowed.* For example, as shown in Figure 1, although G_2 is not a subgraph of G_1, G_2 still can be regarded as matched to G_1 if node skipping or node mismatching is allowed.

Such kind of fuzzy matching is desired in various real applications. For example, the discovery of frequent conserved subgraph patterns from protein interaction networks [1,2] is an important and challenging work in evolutionary and comparative biology, where 'conserved' just means the inexact graph pattern matching allowing node mismatch and node skipping. Similarly, in social network analysis, the direct connection between nodes usually is not the focus; instead, the high-level topological structure with independent paths contracted is of great interest.

* The work was supported by the National Natural Science Foundation of China under Grant No.60303008, No.60673133, No.60703093; the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321905.

J.R. Haritsa, R. Kotagiri, and V. Pudi (Eds.): DASFAA 2008, LNCS 4947, pp. 452–460, 2008.

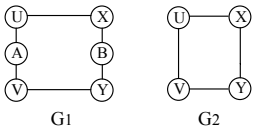


Fig. 1. Inexact Matching

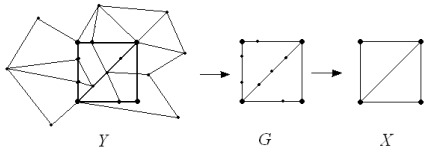


Fig. 2. Topological Minor

Using *Graph Minor* theory [4], the abstract topological structure in many real applications can be described as *topological minor*, and the relation between abstract topological structure and its detailed original graph can be described as *node/vertex disjoint subgraph homeomorphism*. However, to determine whether a pattern graph P is a topological minor of data graph G is non-trivial, and has been proved to be NP-complete when P and G are not fixed [3]. Although Robertson and Seymour [4] have proposed a framework to solve *minor containment* problem that is a generalization of topology containment problem and [5] has implemented the framework, no practically efficient algorithm has been available to solve *ndSHD*, to the best of our knowledge. Here, we propose two algorithms that are based upon state space searching with backtracking. To improve the efficiency, many heuristics have been integrated into the searching procedures to prune the search spaces.

2 Preliminaries

We begin this section with some basic notations. Let $G = (V, E, l)$ be a *vertex labeled graph*, where V is the set of vertices, E is the set of edges and $E \subseteq V \times V$, and l is a label function $l : V \rightarrow L$, giving every vertex a label. The vertex set of G is referred to as $V(G)$, and the edge set is referred to as $E(G)$. A *path* P in a graph is a sequence of vertices v_1,v_2,...,v_k, where $v_i \in V$ and $v_i v_{i+1} \in E$ for each i. The vertices v_1 and v_k are linked by P and are called its *ends*. The number of edges of a path is its *length*, and the path of length k is denoted as P^k. A path is *simple* if its vertices are all distinct. Particularly, a group of paths are *independent* if none of the paths has an inner vertex on another path.

As described in [7], a *topological minor* of a graph is obtained by contracting the independent paths of one of its subgraphs into edges. For example, in Figure 2, X is a topological minor of Y, since X can be obtained by contracting the independent paths of G that is a subgraph of Y.

Formally, as shown in Figure 2, if we replace all the edges of X with independent paths between their ends, so that these paths are *pairwise node independent*, i.e. none of these paths has an inner vertex on another path, then G is a *subdivision* of X. Furthermore, if G is a subgraph of Y, then X is a *topological minor* of Y. As a subdivision of X and a subgraph of Y, if G is obtained by replacing all the edges of X with independent paths with length from l to h, then G is an *(l, h)-subdivision* of X and X is an *(l, h)-topological minor* of Y.

Given two graphs X and Y, if X is a topological minor of Y, then there exists a corresponding *node disjoint subgraph homeomorphism* from X into Y, which is a pair of injective mappings (f, g) from X into Y. Here, f is an injective mapping from vertex set of X into that of Y (all the mapped nodes under mapping f are called *branch nodes* of Y). And g is an injective mapping from edges of X into simple paths of Y such that (1) for each $e(v_1, v_2) \in E(X)$, $g(e)$ is a simple path in Y with $f(v_1)$ and $f(v_2)$ as two ends; and (2) all mapped paths are pairwise independent.

3 Algorithm Framework

3.1 A Rudimentary Algorithm

To determine whether G_1 is an (l, h)-*topological minor* of G_2 is equivalent to find a pair of mappings (f, g) between these two graphs. The final solution of the determination can be described as $\mathcal{M} = (NM, EPM)$, where $NM \subseteq V_1 \times V_2$ is the node match set and $EPM \subseteq E_1 \times (P^l \cup ... \cup P^h)$ is the edge-path match set. All the mapped nodes of G_2 can be denoted by $NM^{(2)}$, and all the mapped paths of G_2 can be denoted by $EPM^{(2)}$.

The process of finding the homeomorphism mapping can be suitably described by means of *State Space Representation* [9]. Each state s of the matching process can be associated with a partial mapping solution $\mathcal{M}_s = (NM_s, EPM_s)$, where NM_s and EPM_s are the node match set and edge-path match set at state s, respectively. Obviously, $\mathcal{M}_s$ contains all the matches we have found so far and will probably be a subset of some final match set $\mathcal{M}$. The algorithm framework based on state space searching is shown as follows.

Algorithm ndSHD1(G_1, G_2, l, h)
Input: G_1,G_2:vertex labeled graphs; l:minimal path length; h:maximal path length.
Output: If G_1 is an (l, h)-topological minor of G_2 return *true* and return *the **first** found node disjoint subgraph homeomorphism* (f, g), otherwise return *false*.

1. Initial(M, R);/*Initialize SHD, generate necessary path information, initialize the basic data structures M and R, which will be described in the following section.*/
2. $s \leftarrow \emptyset$; /*Initialize state as empty state.*/
3. **while** NodeMappingSearch(s,M,R) /*Search in node mapping space. */
4. **if** EdgePathMappingSearch(s,M,R) /*Search in edge-path mapping space.*/
5. **return true**;
6. **return false**;

For example, given two graphs shown in Figure 3, let (l, h) be $(2, 2)$, which means the edges in G_1 can only be mapped to the paths in G_2 with length 2. The running procedure under above parameters is shown in Figure 4. The result of the determination is true, and the node mappings are $NM = \{1-2, 2-8, 3-6, 4-4\}$ and the five edge-path mappings are $EPM = \{12-218, 13-296, 14-234, 23-876, 34-654\}$.

Please note that the answer to the problem is sensitive to the given parameter (l, h). If (l, h) is $(3, 3)$, G_1 is not a topological minor of G_2. The influence of

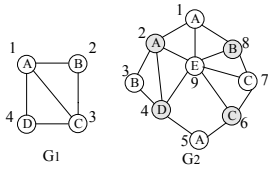


Fig. 3. Running Example

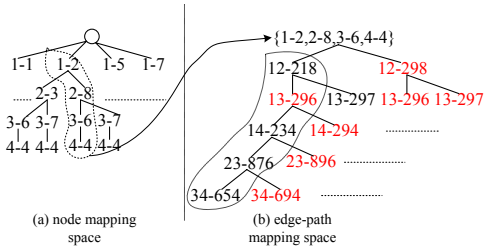


Fig. 4. Two-Level State Space Searching

parameter (l, h) on topology containment determination has been discussed in [8] in detail.

3.2 Basic Data Structures

As described above, we need two basic data structures, one is used to represent the node mapping information; the other is used to represent (l, h) independent path information of G_2. For the former, we use node compatible matrix; the latter, we use independent path matrix as well as a path index structure. Both of them are changing with the transition of the matching state.

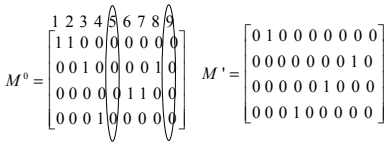


Fig. 5. M^0 and M'

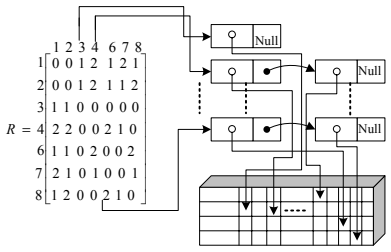


Fig. 6. R and its associated path index structure

We define node compatible matrix $M = [m_{ij}]$ to be an n_1 (rows)$\times n_2$ (columns) matrix whose elements are 1's or 0's, where n_1 and n_2 are the number of nodes in G_1 and G_2, respectively. At the final success state, we can get a final mapping matrix $M' = [m'_{ij}]$ whose elements are 1's or 0's, such that each row contains exactly one 1 and each column contains no more than one 1. The final mapping matrix represents a valid one-to-one mapping between nodes of G_1 and G_2, while the initial compatible matrix M^0 represents the probable mappings between nodes of G_1 and G_2. Obviously, for each element m'_{ij} of M', $(m'_{ij} = 1) \rightarrow (m^0_{ij} = 1)$.

Clearly, reducing the number of 1's in M is the key to reduce the node mapping space, which is the basic idea of Lemma 1. When constructing independent path matrix and its associated path index structure, we only need to enumerate all the (l, h)-paths between all *candidate branch node* pairs, which is based on Lemma 2.

Lemma 1. *The number of elements of the independent path set starting from a vertex v is no more than $d(v)$, where $d(v)$ denotes the degree of v.*

Lemma 2. *If G_1 is an (l,h)-topological minor of G_2 under subgraph homeomorphism (f,g), then $g(E_1)$ only contains paths ending with those branch nodes in G_2.*

Then, we can define the independent path matrix $R = [r_{ij}]$ to be an $n_2' \times n_2'$ matrix (n_2' is the number of candidate branch nodes in G_2) with r_{ij} representing the number of (l,h) paths between the node pair (v_i, v_j) in G_2. The detailed path information is stored in an array of lists *RLists*, where each list contains path addresses that point to the physical storage of the paths for some r_{ij}.

Figure 5 and Figure 6 show these two basic data structures used in the running case shown in Figure 3.

3.3 State Space Searching

The procedure of node mapping space searching and edge-path mapping space searching are similar to each other. These two procedures are shown as follows.

Algorithm Node/EdgePathMappingSearch1 (s,M,R)
Input: s:the current matching state; M:the current node compatible matrix; R: the current independent path matrix.
Output: *found*: a boolean variable indicating whether a complete node/edge-path mapping has been found.

```
1. if(s is dead state) return false;
2. if(s is complete mapping state) return true;
3. let found←false
4. while(not found && Exists Valid node/edge-path Mapping Pair)
5.    m ←GetNextNodePair(); /*m ←GetNextEdgePathPair();*/
6.    s' ←BackupState(s);
7.    NM_s ← NM_s ∪ {m}; /*EPM_s ← EPM_s ∪ {m}*/
8.    Refine(M,R);
9.    found←Node/EdgePathMappingSearch(s, M, R);
10.   if(found) return true;
11.   s ←RecoverState(s');
12. return false;
```

From lines 1-2, we can see that when a new state s arrives, s can be a *dead state* or *success state*(complete mapping state). The state space search arrives at a *success state* if all the node mappings or edge-path mappings have been found, which means $|NM_s| = |V_1|$ or $|EPM_s| = |E_1|$. The node mapping state space search arrives at a *dead state* if there is a row with all 0's in node compatible matrix M of the current state, i.e. $\exists i, |NM_s| \leq i \leq n_1$,s.t. $\sum_{1 \leq j \leq n_2} m_{ij} = 0$. And the edge-path mapping state space search arrives at a *dead state* if there is no path between some pair of branch nodes, i.e., $\exists i, |EPM_s| \leq i \leq n_2'$,s.t. $\prod_{(f^{-1}(node(i)), f^{-1}(node(j))) \in E_1} r_{ij} = 0$, where $node(i)$ gets the vertex corresponding to the i-th column in matrix R.

3.4 Refinement Procedure

To traverse all possible mapping branches is time-consuming, so space pruning is essential for ndSHD. For this purpose, we devise two refinement procedures on R and M, respectively. Lemma 3 and 4 show the correctness of refinement on R, and Lemma 5 shows the correctness of refinement on M.

Lemma 3. *In the matching process, let s be the current state, if $v \in NM_s^{(2)}$ and $\mathcal{M}_s$ will be a partial solution of some final solution $\mathcal{M}$, then any path with v as inner vertex will not $\in EPM^{(2)}$.*

Lemma 4. *In the matching process, let s be the current state, if $p \in EPM_s^{(2)}$ and $\mathcal{M}_s$ will be a partial solution of some final solution $\mathcal{M}$, then any path passing trough the inner vertex of p will not $\in EPM^{(2)}$.*

Lemma 5. *In the matching process, let s be the current state, if $(v_i, v_j) \in NM(v_i \in V_1, v_j \in V_2)$ and $\mathcal{M}_s$ will be a partial solution of some final solution $\mathcal{M}$, then the following statements hold true:*

1. *$\prod r_{j'k} > 0$, where $j' = Index(v_j), k \in Index(V)$ and $V = \{v_2 | v_1 \in Adjacent(v_i) \wedge (v_1, v_2) \in NM_s\}$.*
2. *$\forall v' \in V', \exists v \in V_2$ such that $l_2(v) = l_1(v')$ and $r_{j'k} > 0$, where $j' = Index(v_j)$, $k = Index(v)$ and $V' = \{v' | v' \in Adjacent(v_i) \cap (V_1 - NM_s^{(1)})\}$.*
3. *The path set consisting of the paths to which all mentioned $r_{j'k}$'s in (1) and (2) indicate is independent.*

In the above statements, the function $Index(v)$ gets an index in R for a node v in G_2; and $Adjacent(v)$ obtains the adjacent vertex set of v.

3.5 More Efficient Searching Strategy

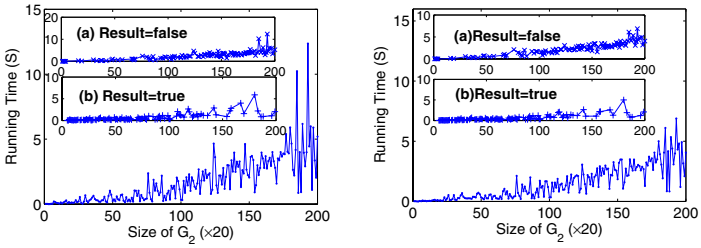
A basic observation of the above refinement procedures is that the constraint resulting from an edge-path match will be more restrictive than that resulting from a node match. Hence, a better strategy is to try edge-path match as early as possible, rather than performing edge-path match until complete node match has been found. We denote these two strategy as s_1 (old strategy) and s_2(new strategy), respectively; and algorithms employing two strategies are denoted as *ndSHD1* and *ndSHD2*, respectively. Intuitively, in *ndSHD2* the searching procedure will meet with the dead state very early if the current searching path will not lead to a successful mapping solution, thus the searching procedure will fast backtrack to try another mapping solution.

The framework of algorithms *ndSHD2* and *Node/EdgePathMappingSearch2* are similar to that of *ndSHD1* and *Node/EdgePathMappingSearch1*, and thus the details are omitted here due to the space limitation.

4 Experimental Evaluation

To test the efficiency of the algorithms, we generate the synthetic data sets according to the random graph [10] model that links each node pair by probability p. All generated graphs are vertex labeled undirected connected graphs. We also randomly label vertices so that the vertex labels are uniformly distributed. We implement the algorithm in C++, and carry out our experiments on a Windows 2003 server machine with Intel 2GHz CPU and 1G main memory.

The efficiency of the algorithms is influenced by the following factors: N_1: node size of G_1, N_2: node size of G_2, M_1: average degree of G_1, M_2: average degree of G_2, (L, H): the minimal and maximal path length. The efficiency also can be influenced by the number of vertex labels.

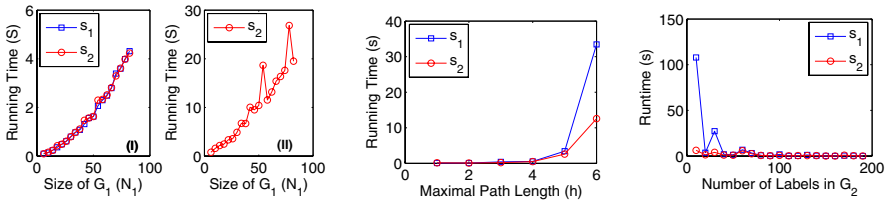


(a) $ndSHD1$ (b) $ndSHD2$

Fig. 7. Efficiency and scalability with respect to the growth of size of data graph (G_2). The inset of (a), (b) show runtime of all running cases where the determination result is *true*, *false* respectively.

First we will demonstrate the scalability with respect to the growth of the size of nodes of data graph G_2. We use a complete graph with 4 uniquely labeled nodes as a minor graph; we generate overall 200 data graphs G_2 with node size varying from 20 to 4000 in increment of 20. The average degree of each data graph is fixed as 4 and nodes of each graph are randomly labeled as one of overall 20 labels. L and H are fixed as 1 and 3, respectively, meaning that the path length is in the range of $[1, 3]$. Thus the parameters can be denoted as $N_1 4 M_1 3 M_2 4 L1 H3$. From Figure 7, we can see that *ndSHD1* and *ndSHD2* both are approximately linearly scalable with respect to the number of nodes in G_2, irrespective of the result of the determination.

Next we will show the scalability of *ndSHD1* and *ndSHD2* with respect to the size of G_1. We fix some parameters as $M_1 4 N_2 4 k L1 H3$ and vary the size of G_1 from 6 to 82 in increment of 4 to generate 20 minor graphs. Each minor graph is uniquely labeled. Two data graphs are used, one has average degree M_2 as 8 and the other as 20. These two data graphs are randomly labeled as one of 200 labels. Figure 8(a)(I) ($M_2 = 8$) and (II) ($M_2 = 20$) show the results. As can be seen, *ndSHD1* and *ndSHD2* both are approximately linearly scalable with respect to the number of nodes in G_1. In (II), running time of *ndSHD1* is not



(a) Size of G_1 (b) Maximal path length (c) Label number in G_2

Fig. 8. Scalability of two algorithms

available when $M_2 = 20$, meaning that all running cases need time larger than one hour.

Figure 8(b) shows the runtime of the algorithm with respect to (l, h). Parameters of this experiment are set as $N_1 4 M_1 3 N_2 1 k M_2 8 L 1$. The minor graph is uniquely labeled; data graphs are randomly labeled as one of 20 labels. As can been seen, the broader the range is, the longer the running time is; and the runtime of *ndSHD1* and *ndSHD2* both increase dramatically with the growth of upper bound of path length. However, the increasing speed of *ndSHD2* is slower than that of *ndSHD1*, which implies that *ndSHD2* is more efficient than *ndSHD1* with respect to larger h. The super linearly growth of the runtime with the increase of upper bound of the path length can be partly attributed to the exponential growth of the number of potentially mapped paths. However, in various real applications, larger upper bound of path length is non-meaningful when performing fuzzy matching on graph data, and usually upper bounds less than 3 are desired.

To examine the impact of the number of vertex labels on the performance of *ndSHD1* and *ndSHD2*, we use a uniquely labeled graph with 6 nodes and 15 edges as minor graph, a graph with 1000 nodes and 4000 edges as data graph. We randomly labeled the data graph from 10 labels to 200 labels in increment of 10 to generate 20 different labeled data graphs. L and H are set as 1 and 3, respectively. The result of this experiment is shown in Figure 8(c). Clearly, runtime of *ndSHD1* and *ndSHD2* substantially decrease with the growth of number of labels of G_2, which confirms to what we have expected, since larger number of labels in G_2 can reduce the node mapping space between minor graph and data graph. We also can see that *ndSHD2* outperforms *ndSHD1* to a great extent when the number of labels is small.

5 Conclusions

In this paper, we investigated the problem known as node disjoint subgraph homeomorphism determination; and proposed two practical algorithms to address this problem, where many efficient heuristics have been exploited to prune the futile searching space. The experimental results on synthetic data sets show that our algorithms are scalable and efficient. To the best of our knowledge, no practical algorithm is available to solve node disjoint subgraph homeomorphism determination.

References

1. Kelley, R.B., et al.: Conserved pathways within bacteria and yeast as revealed by global protein network alignment. PNAS 100(20), 11394–11399 (2003)
2. Sharan, R., et al.: Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. In: RECOMB 2004, pp. 282–289 (2004)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-completeness. W.H. Freeman and Company, New York (2003)
4. Robertson, N., Seymour, P.D.: Graph minors. XIII: The disjoint paths problem. Journal of Combinatorial Theory 63, 65–110 (1995)
5. IIIya, V.: Hicks:Branch Decompositions and Minor Containment. Networks 43(1), 1–9 (2004)
6. Ullmann, J.R.: An Algorithm for Subgraph Isomorphism. Journal of the ACM 23, 31–42 (1976)
7. Diestel, R.: Graph Theory. Springer, Heidelberg (2000)
8. Jin, R., Wang, C., Polshakov, D., Parthasarathy, S., Agrawal, G.: Discovering frequent topological structures from graph datasets. In: KDD 2005, Chicago,USA, pp. 606–611 (2005)
9. Nilsson, N.J.: Principles of Artificial Intelligence. Springer, Heidelberg (1982)
10. Erdös, P., Rényi, A.: On random graphs. Publicationes Mathematicae, 290–297 (1959)