Mining Conserved Topological Structures from Large Protein-Protein Interaction Networks*

Yanghua Xiao+ Wei Wang+ Wentao Wu+
+ Fudan University, China{Shawyanghua, wentaowu1984,Vincent.wzb}@gmail.com,weiwang1@fudan.edu.cn

ABSTRACT

Analysis of Protein-Protein Interaction (PPI) networks is of great significance in evolutionary biology. Because of high computation cost, recently *multi-PPI network alignment* becomes hot topic. In this paper, we proposed *conserved topological structures mining* based *multi-PPI network alignment* technology. The most challenging problems in conserved topological structure mining are the large size of the real PPI networks and the requirement of inexact alignment of networks. To solve this problem, we develop an algorithm framework: Conserved Substructure Mining (CSMiner) for PPI networks. In the algorithm, we synthesize many techniques to boost the mining procedure, including a novel efficient pattern growth method, an efficient node disjoint subgraph homeomorphism determination algorithm and the integrated strategy of these two algorithms. We also demonstrate the efficiency and effectivity through the experiments on real PPI networks of *Saccharomyces Cerevisiae* and *Drosophila Melanogaster.*

1. Introduction

In evolutionary and comparative biology, PPI network analysis has attracted more and more research attentions. [9][10] focused on finding motifs in large PPI networks, and these network motifs can be employed to explore the interactomes. [11][12][13] focused on aligning multi-PPI networks to find conserved structures, such as paths[11], clique-like structures[12] or general structures[13], and these conserved structures can be used to analysis the relevant functionality across species. Bearing in mind the importance of PPI network analysis, we must also be aware that neither motif finding nor network alignment is a trivial or easy thing, especially the latter.

In multi-PPI network alignment, we need to find the conserved subnetworks within the PPI networks belonging to different species. The conservation is measured in terms of sequence similarity and network topology similarity. The challenging in PPI network alignment are:

- The PPI network data contains many noisy data. Hence, alignment of multi-PPI networks in the real application often needs to allow *node skipping* or *node mismatching*. It increase the computation cost of graph matching.
- The PPI network data is relatively *large and complex*. More and more PPI networks which have more than several thousand nodes and ten thousands edges have been discovered. Moreover, the PPI network follows many well-known laws of complex networks, such as 'scale free' property.

Figure 1 shows the PPI network of *Saccharomyces Cerevisiae*.

From the graph mining perspective, multi PPI network alignment is considered as finding the frequent conserved subgraph patterns from multi PPI networks. These conserved sub-networks are similar to each other with relaxation of node mismatch and node skipping. Hence, the task is equivalent to find those *conserved topological structures*, conserved across species with some noisy node skipped or edges contracted.

Although many graph mining algorithm frameworks have proposed, they can't be easily extended to facilitate the accomplishment of PPI network alignment for the following reasons. First, most of the existing graph mining systems[1-8] focused on mining subgraph patterns from relatively *small and simple* graphs with size less than one hundred nodes and a very little average node degrees. However, the PPI network data is *much larger and more complex*. Second, traditional graph mining systems only need to mine out precise subgraph patterns that are subgraph isomorphic to data graphs. On the contrary, the focus of PPI mining is not the precise pattern, but the fuzzy pattern allowing node mismatching and skipping. The process of fuzzy patterns mining is based on the subgraph homeomorphism determination (SHD) which is more difficult than subgraph isomorphism determination (SID).

To solve these problems, we develop an algorithm framework: Conserved Substructure Mining (CSMiner) for PPI networks. Besides the algorithm framework CSMiner, we also make the following contributions.

- We proposed the *conserved topological structures minging based multi-PPI network alignment* technology.
- We propose an efficient *node disjoint subgraph homeomorphism determination algorithm* to perform the inexact pattern matching in multi-PPI network alignment
- We devise the novel *pattern growth operators* to enumerate patterns completely and compactly. We also propose a strategy to treat the pattern matching procedure as a white box and integrate it into the whole mining procedure, to boost the mining procedure.

2. Preliminaries

In this section, we will explain some basic notations and concepts. Then we will give the formal definition of the problem.

2.1 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

*Supported by the National Natural Science Foundation of China under Grant No.60303008 and 60373133; the National Grand Fundamental Research 973 Program of China under Grant No. 2005CB321905

function $l:V \rightarrow L$, assigning to every vertex a label. The vertex set of G is referred to as $V(G)$, and its edges set as $E(G)$. A *path P* in a graph is a sequence of vertices $v_1,v_2,\ldots,v_k$, where $v_i \in V$ and $v_iv_{i+1} \in E$. 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 is *independent* if none of the paths has an inner vertex on another path. In the other words, a path intersecting with other paths only at its ends can be called as an *independent path*.

2.2 Inexact Graph Matching

The existing inexact graph matching is considered as *subgraph isomorphism* between graphs. However, in multi PPI network alignment, subgraph isomorphism cannot represent the fuzzy matching in the sense of topological structure, which will be illustrated in the example 1.

Example 1. As shown in Figure 2, G_2 is not a subgraph of G_1 nor G_3, in subgraph isomorphism based alignment strategy, such as *maximal common subgraph*[15]. However, from the viewpoint of abstract topological structure, G_2 matched to G_1 with relaxation of node skipping or node mismatching. In other words, G_2 matched to G_1, because G_2 retains the topological structure of G_1 by contracting the paths in G_1 into the corresponding edges in G_2. Similarly, G_2 also retains the topological structure of G_3. As a result, G_2 is considered as a conserved pattern when performing pairwise alignment between G_1 and G_3.

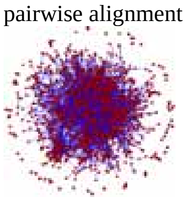


Figure 1:PPI Network

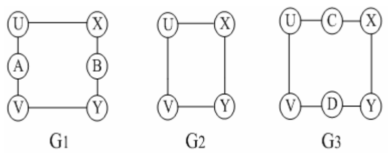


Figure 2:Inexact Graph Matching

From the perspective of *Graph Minor* theory [19], the *abstracted topological structure* in many real applications can be described as *topology minor* which is an abstraction that focuses on the abstract structure of the graph, and the relation between topological structure and its original graph can be described as *node/edge disjoint subgraph homeomorphism*.

2.3 Topology Minor

A topology minor of a graph is generated by contracting the independent paths of one of its subgraphs into edges. For example, in Figure 3, X is a topology minor of Y, since X is generated by contracting the independent paths of Y's subgraph: G. Clearly, contracting independent paths helps simplify a (sub)graph without compromising its topological information.

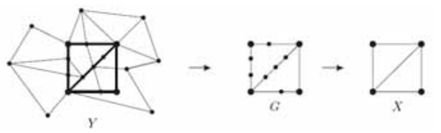


Figure 3: Topology Minor

Formally, as shown in Figure 3, if we replace all the edges of X with independent paths between their ends, so that these paths are *pair-wise node independent*, *i.e.* none of these paths has an inner vertex on another path, then G is a *subdivision* of X, denoted as $T(X)$. If G is a subgraph of Y, then X is a *topology minor* of Y. As a subdivision as 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 a *(l, h)-subdivision* of X and T is a *(l, h)-Topology Minor* of Y.

If X is a topology minor of Y, then all the edges of X can be mapped to a simple path of Y, all the nodes in X can be one to one mapped to nodes of Y, and the mapped nodes are called *branch nodes* of Y.

The generalization of topology minor is *minor*[17], which is obtained by contracting some edges of one of its subgraphs. If X is a minor of Y, $T(X)$ may not be a subgraph of Y, but if X is a topology minor of Y, $T(X)$ must be a subgraph of Y. So topology minor is the most appropriate concept to represent the abstracted or simplified topological structure.

2.4 Node Disjoint Subgraph Homeomorphism

From viewpoint of relation between graphs, minor, topology minor as well as subgraph correspond to three basic relationships between graphs: *subgraph homeomorphism*, *node disjoint subgraph homeomorphism* and *subgraph isomorphism*.

Subgraph isomorphism from P into G is an injective mapping of vertices and edges from P into that of G. The *subgraph homeomorphism* from P into G is a pair of injective mappings (f, g), the first from vertices of P into G, and from edges of P into simple paths of G. In addition, if P is subgraph homeomorphic to G and all the mapped paths of G are node or edge disjoint, *i.e.* all the mapped paths are pairwise independent, then P is a topology minor of G and P is *node or edge disjoint subgraph homeomorphic* to G.

In some applications, such as multi-PPI network alignment, extracting abstracted topological structure is very useful. Node disjoint subgraph homeomorphism is more *flexible* than subgraph isomorphism and more *restricted* than general subgraph homeomorphism. As a result, more *covert* and *meaningful* conserved patterns will be found. The relation of three basic graph relationship is shown in Figure 4.

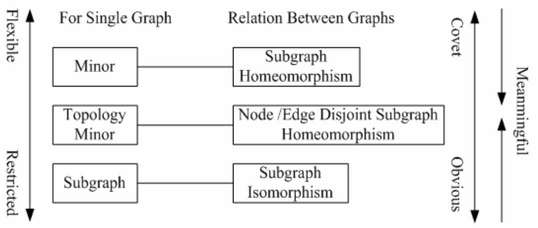


Figure 4: Relation among Three Basic Graph Relationship.

2.5 Problem Definition

With the concept of topology minor at hands, we can formally give the definition of the problem.

Definition 1 (*Conserved Topological Structures*). For a given set ***D*** consisting of k PPI networks, parameters l and h, the *Conserved Topological Substructures* is the graph that is isomorphic to a *(l, h)-Topology Minor* of each PPI network in ***D***.

For a given k PPI networks set ***D***, parameters l and h, the problem of *Mining Conserved Topological Substructures from PPI networks* is to find all the graph that is isomorphic to a *(l, h)-Topology Minor* of each PPI network in ***D***. For the convenience of discussion, this problem is denoted as $P(\boldsymbol{D}, k, l, h)$.

Obviously, from definition 1, we can see that *Mining Conserved Topological Substructures from PPI networks* is similar to traditional frequent graph pattern mining. However, there are still two major distinctive features. The first is the frequency of the intended pattern is 100% in the problem of *Mining Conserved Topological Substructures from PPI networks*, due to its origination from problem of network alignment. The second is the graph relation utilize different criteria of pattern matching, node disjoint subgraph homeomorphism is the substitution for subgraph isomorphism.

Clearly, topology minor is the essential of the whole problem, therefore from this perspective, this problem also can be called as *mining frequent (l,h)- topology minor from PPI networks.*

In the problem of *Mining Conserved Topological Substructures from PPI networks,* the *anti-monotone* property also holds. In the context of conserved topological structure, the anti-monotony implies that if a given graph G is a conserved topological structure of the PPI networks, then any graph subgraph isomorphic to G must be also conserved pattern. As a result, all G's subgraphs are trivial in the sense that these conserved substructures can be inferred from the G. Hence, in multi PPI network alignment, we can only take care of maximal conserved graphs.

Lemma 1 (*Closure of Conserved Topological Structure Space under subgraph isomorphic relation*). For a given problem $P(\boldsymbol{D}, k, l, h)$, let *Conserved Topological Structure Space* be the collection of all the conserved topological structures of the problem, denoted as $\boldsymbol{M}$. Then $\boldsymbol{M}$ is *closed under subgraph isomorphic* relation, which means that if graph $g\in\boldsymbol{M}$, then for any g' subgraph isomorphic to g, $g'\in\boldsymbol{M}$.

Property 1 (*Anti-monotony of the conservation of Conserved Topological Structures under subgraph isomorphic relation*). For a given problem $P(\boldsymbol{D}, k, l, h)$, if G is *conserved*, then any graph subgraph isomorphic to G is also *conserved*.

Property 1 can also be expressed reversely, if G is not conserved, then any G's supergraph will not be conserved, which often called as *downward closure property of G.* This property can be used to prune the candidate pattern space

3. Algorithm Framework

In this section, we first sketch out the Apriori-based algorithm framework.

3.1 Apriori-based Conserved Topological Structure Mining

As a general algorithm framework, level-wise search based on Apriori property has been widely used to enumerate candidate patterns in frequent graph pattern mining [1][2]. In this paper, we also utilize the Apriori based level-wise search as the algorithm framework, which is shown in Algorithm 1 and the notations used in Algorithm 1 is shown in Table 1.

Algorithm 1 Apriori-CSMiner (*D*, *l*, *h*)

```
Input: PPI networks $\boldsymbol{D}$, $l$ :minimal path length, $h$ :maximal path length;
Output: conserved topological structures $\boldsymbol{F}$
Method:
      //phase 1: Initial Mining
1. $\boldsymbol{M} \leftarrow VertexMappingBuilding(\boldsymbol{D})$
2. Initia Mining(D) ; //Initialize the mining procedure.
3. $F^1 \leftarrow \{e \mid \forall G\in\boldsymbol{D}, e\in E(G)\}$
4. $k\leftarrow 1$
5. while $F^k\neq\emptyset$
      //phase 2: Pattern Growth
6.    $C^{k+1}\leftarrow$CandidateGen($F^k$)
8.    $F^{k+1}\leftarrow C^{k+1}$
   //phase 3: Pattern Matching Validation
9.    for each candidate $P^{k+1}\in C^{k+1}$
10.       for each $G\in\boldsymbol{D}$
11.           if $ndSHD(P^{k+1}, G)=false$ then
12.               $F^{k+1}\leftarrow F^{k+1}- \{P^{k+1}\}$
13.   $k\leftarrow k+1$
14. return $\boldsymbol{F}=\{F^1,\ldots, F^k\}$
```

Table 1 Notations in Algorithm 1

Notation	Description
M	The set of Initial Node Compatible Matrix
F^k	Frequent pattern set of size *k*
C^k	Candidate pattern set of size *k*
P^k	A pattern with size k

As illustrated in Algorithm 1, the entire procedure can be divided into three major phases, *initial mining*, *pattern growth* and *graph matching validation*.

1. *Initial Mining*. To correlate all input networks, vertex mapping is build. The details will be discussed in section 3.2. In this phase, we also generate all the paths with length between l and h for each PPI network, as well as an index structure, which will be discussed in 4.2 in detail. In this index structure, all of the nodes and the paths in the data graph, are indexed. This index structure will be employed to speed up the procedure of node disjoint subgraph homeomorphism based pattern matching.

2. *Pattern Growth(Enumeration)*. In the pattern enumeration phase (lines 6-8), we employ *level-wise Pattern Growth* as the enumeration strategy. To ensure the completeness and compactness of pattern enumeration, three graph pattern join operators have been devised based on the linear order defined on the patterns set: one *Forward Join operator*, two *Backward Join Operator*, which will be discussed in section 5.

3. *Pattern matching* .In this phase (line 9-11), node disjoint SHD is performed. It is the most crucial part of the whole algorithm. The basic idea is to employ depth first search in the state space with backtrack. For SHD problems, node mapping space and edge-path mapping space will be searched. In CSMiner, first we look for a valid node mapping, and then explore the edge-path space to find a valid edge-path mapping. In this edge-path mapping, all the paths need to be pair-wise independent. Once a node is selected or a path is selected, all the paths passing through the node or joint with the path are marked as negative and cannot be used as a mapped path. The details will be introduced in section 4. Furthermore, in this phase, when we are going to perform a ndSHD between pattern graph G_1 and data graph G_2, it's not necessary to start the from scratch, we only need to start from the matched state when performing ndSHD between one of G_1's parent pattern and G_2 .This process will be described in section 7.

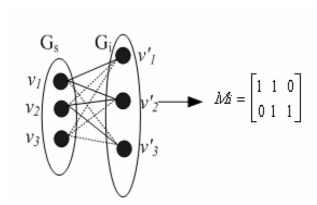
3.2 Vertex Mapping Building

The first conservation level in multi-PPI Networks alignment is protein sequence similarity. Hence, before we perform topology structure alignment, need to accomplish the pairwise sequence alignment. Luckily, this preliminary step can be achieved by using BLAST[18], a well-known sequence alignment tool.

After protein sequences alignment have been performed, we will find the similar protein sequence pair from different species. The similarity score value will be larger than a similarity threshold α. Then we can build the mapping relation between each pair of PPI networks in terms of α.

Procedure of Vertex Mapping Building. Formally, for the problem $P(\boldsymbol{D}, k, l, h)$, first, we select a *seed network* G_s such that $G_s = arg\ min\{|V(G)| G \in \boldsymbol{D}\}$. Then for each $G_i \in \boldsymbol{D}$, we build a *partial multi-value mapping* from $V(G_s)$ to $V(G_i)$ in terms of the protein sequence similarity threshold α. We only keep the mapping relations whose similarity score larger than α. We use $V_s \subseteq V(G_s)$ to denote the nodes participating in the mapping in $V(G_s)$. And the partial mapping between G_s and $G_i \in \boldsymbol{D}$ is denoted as $f_i : V_s \rightarrow V(G_i)$. Overall, we have a mapping set $F=\{f_1, \ldots, f_k\}$. Then for each $f_i \in F$, we define *node compatible matrix* $M_i=[m_{pq}]$ to be n_1 (rows) $\times n_2$ (columns) matrix whose elements are 1's or 0's and $m_{pq}=1$ *iff* $v_q \in f_i(v_p)$, where $n_1=|V_s|$ and $n_2=|f_i(V_s)|$.

Example 2. Figure 5 shows an example of the mapping relation in terms of protein sequence similarity between G_s and G_i. The solid line represent the mapping relation with similarity value larger than α, the dotted line represent that less than α. Note that $v_3 \notin V_s$, as no protein sequence of G_i is similar to v_3. Therefore, the mapping from $V(G_s)$ to $V(G_i)$ is only a *partial* mapping. Moreover, it is not difficult to see that the mapping also is a *multi-value* mapping. M_i is the *node compatible matrix*.



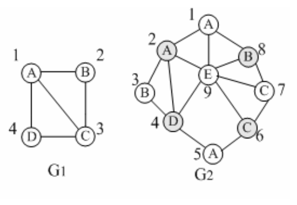


Figure 5: Mapping Relation . Figure 6: Running Example

In the above vertex mapping building procedure, we need to have a deep look on the *partial mapping* property of f_i. f_i is only a *partial mapping*, which implies that there exist some vertices of *seed network*, for which no mapping can be built. The proteins belong to $V(G_s)$-V_s will not occur in the conserved topological structures as branch nodes. α is the key parameter in the algorithm. It can be determined by the distribution of similarities scores between proteins of different species.

The vertex mapping building procedure is also a vertex relabeling procedure. The node with similar sequence will share the same label. It will make the process of node mapping simpler. Node Disjoint Subgraph Homeomorphism Determination(ndSHD)

The core of CSMiner is *node disjoint Subgraph Homeomorphism Determination (ndSHD)*. In the Apriori-CSMiner framework, given a candidate pattern P and PPI networks $\boldsymbol{D}$, we need to determine whether P is isomorphic to a (l, h)-topology minor of each $G \in \mathbf{D}$.

To simplify the description, we give some notations. Assume that the pattern graph $G_1=(V_1,E_1,l_1)$ is a (l, h)-topology minor of PPI network $G_2=(V_2,E_2,l_2)$ under the node disjoint subgraph homeomorphism (f, g), where $f: V_1 \rightarrow V_2$ and $g: E_1 \rightarrow P^l \cup \ldots \cup P^h$. The number of vertices and edges of G_1 and G_2 are n_1, m_1 and n_2, m_2, respectively.

3.1 Two-level State Space Searching

To determine whether G_1 is a (l, h) topology minor of G_2 is equivalent to find a pair of mapping (NM, EPM) between these two graphs. The mapping $NM \subseteq V_1 \times V$ maps the nodes in G_1 to the nodes with the same label in G_2. While $EPM \subseteq E_1 \times P^l \cup \ldots \cup P^h$ map all the edges of G_1 to the path with the same end in G_2. All the mapped nodes of G_2 is denoted as $NM^{(2)}$, and all the mapped paths of G_2 is denoted as $EPM^{(2)}$.

The process of finding the homeomorphism mapping can be described by means of *State Space Representation* [16]. Each state s of the matching process can be associated to a partial mapping solution $M(s)$, which contains all the matches we have found and is a subset of the final match set.

In terms of the above discussion, it's naturally to employ the two-level state space searching as the framework of ndSHD. The algorithm framework is shown in Algorithm 2. At first, we initialize two basic data structures of the algorithm: *node compatible matrix M* and *independent path matrix R* as well as its associated *path indexed structure*. Then we start the node matching process from the empty state (line 3). Each time we select a branch in the state space, a state s transits to a new succeeding state s' by adding a new match, which is a node pair or an edge path pair, to the partial solution. Each time a new match state arrives, M and R are updated so that the node mapping space and edge-path mapping space can be pruned. When a complete node mapping has been found (line 5), the matching process will come to the second level: edge-path matching space search (line 6). Similar to the search process in node mapping space, each time a branch is selected, an edge-path pair is added to the partial mapping solution and the independent path matrix is updated. The process continues until a complete edge-path mapping is found. If all the possible valid branches in the subspace rooted at current state s have been explored, the searching process *backtracks* to the parent state of s. And any time the procedure enters into *dead state* which will be discussed in 4.3, the whole process will stop and return false.

Algorithm 2 NodeDisjointSHD (G_1, G_2, M_{12}, l, h)

Input: G_1, G_2: vertex labeled graphs, M_{12}: the initial node compatible matrix, l :minimal path length, h :maximal path length ;

Output: If G_1 is a (l, h) topology minor of G_2 return true and return the first found node disjoint subgraph homeomorphism (f, g), otherwise return false.

Method:

```
// Initialize the basic data structures
1. Initial(M, M_12)
2. Initial(R)
3. s←∅ //initialize state as empty state
4. s←NodeMappingSearch(s,M,R) //node mapping space search
5. If not IsValid(s) return false else
6.    s ←EdgePathMappingSearch(s,M,R)
7. If not IsValid(s) return false else return true
```

Example3. Given the two vertex labeled graphs as shown in Figure 6, the two-level state space searching procedure for a (2,2) topological mapping is shown as Figure 7.

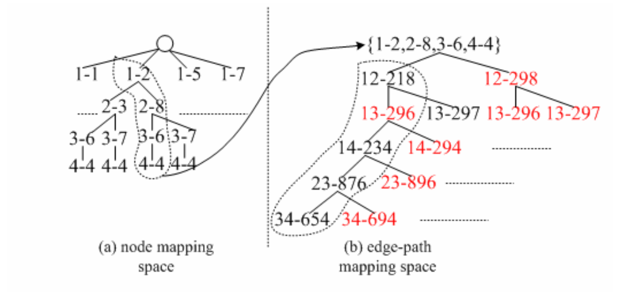


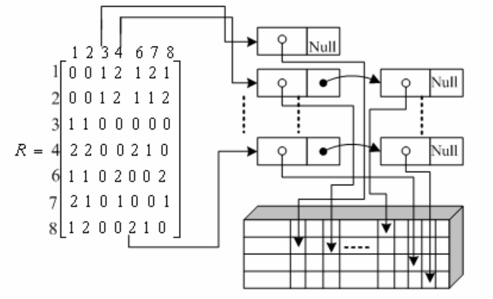
Figure 7: Two Level State Space Searching

3.2 Basic Data Structure

As described in 4.1, there are two basic data structures. Node compatible matrix is used to represent the node mapping information. Independent path matrix is used to represent (*l*, *h*) independent path information of G_2. Both of these two data structures are changing with the transition of the matching state.

$$M^0 = \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \left[\begin{array}{ccccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array} \quad M' = \left[\begin{array}{ccccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Figure 8: M^0 and M'



$$R = \begin{array}{c|c} & \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 6 & 7 & 8 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 6 \\ 7 \\ 8 \end{array} & \left[\begin{array}{ccccccc} 0 & 0 & 1 & 2 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 & 1 & 1 & 2 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 2 & 1 & 0 \\ 1 & 1 & 0 & 2 & 0 & 0 & 2 \\ 2 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 2 & 0 & 0 & 2 & 1 & 0 \end{array}\right] \end{array}$$

Figure 9: R and its associated Path Indexed Structure

The elements of node compatible matrix $M=[m_{ij}]$ to be n_1 (rows) $\times n_2$ (columns) matrix are 1's or 0's. In the final mapping matrix $M'=[m'_{ij}]$, each row contains exactly one 1 and each column contains no more than one 1. The final mapping matrix represents a one to one mapping between nodes of G_1 and G_2, while the initial compatible matrix represents the probable mappings. The two matrixes of Figure 6 are shown in Figure 8. Obviously it follows that M', ($m'_{ij}=1$)→($m^0_{ij}=1$).

LEMMA 2: The number of elements of the independent path set starting from a specified vertex $v_i \in V$ is no more than $d(v_i)$, where $d(v_i)$ denotes the degree of v_i.

Because every path set starting from v_i must pass through one or more edges incident with v_i. So the independent path set starting from v_i has at most $d(v_i)$ elements. According to lemma 2, the node *v* in G_1 cannot be mapped to those nodes in G_2 whose degree is less than $d(v)$. Therefore, we refined the initial compatible matrix M^0 in accordance with the following rule: $m^0_{ij}=0$ *if* $d(v_i)>d(v_j)$. As shown in Figure 3, v_1 in G_1 cannot mapped to v_5 of G_2, although these two nodes have the same label.

When constructing *Independent Path Matrix* and its associated *Path Indexed Structure*, whether we need to generate all the (*l*, *h*) path information of G_2. The answer is false.

LEMMA 3: If G_1 is a (*l* , *h*)-topology minor of G_2 under subgraph homeomorphism (*f* , *g*), then gE_1 only contains paths ending with those branch nodes in G_2.

From lemma 3, we only need to generate all the (*l*, *h*) paths between any *candidate branch node* pairs. These candidate branch nodes have been filtered out by matrix M^0. For example, in Figure 8 shown, column 5 and 9 have only 0's, then v_5 and v_9 in G_2 cannot be branch nodes, then all paths in figure 3 starting from v_5 and v_9 can be ignored. The cardinality of the candidate branch node set is denoted as n_2'.

The independent path matrix $R=[r_{ij}]$ is n_2' (rows) $\times n_2'$(columns) matrix. Its elements are positive integers or 0's, which represent the number of (*l*, *h*) paths between the node pair (v_i,v_j) in G_2. We associate *R* to a path index structure RLists, and every list in RLists contains all the path addresses that point to the physical storage of the path.

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 in Algorithm 3.

From line 1-2, when a new state *s* arrives, *s* can be a *dead state* or *success 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*. While the edge-path mapping state space search arrives at a *dead state* if *there is no path between any branch node pairs*, which can be determined from independent path matrix *R* of the current state.

From The 4-12 lines, when search process enters success state or dead state, the procedure is over. If success state arrives, the complete mapping is found and the procedure returns true. If dead state arrives, the procedure returns false. For other cases, the procedure will continue exploring the state space.

Assume the process comes to state *s* whose *M(s)* is a partial solution. Then as long as a valid mapping pair exists, which may be a node pair or an edge-path pair, we need to generate a new state by adding the new match (line 7). To enable backtracking, we need to backup the current state first (line 6). Under the influence of the new added match, the node compatible matrix and independent path matrix need to be updated (line 8,9). Then DFS continues, until the search enters into dead state or success state. If we cannot find a success state in subtree space rooted at *s*, we recover the state *s* (line 12), and try the sibling state branch.

Algorithm 3 Node/EdgePatbMappingSearch (*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 indicate whether a complete node/edge-path mapping has been found
Method:

```
1. if s is dead state then return false;
```

```
2. if s is complete mapping state then return true;
3. found←false;
4. while(!found && Exists Valid Node/edge-path Mapping Pair)
5.    m←nextValidPair()   // generate a match m;
6.    s'←BackupState(s)
7.    NM(s)←NM(s)∪{m}  // or EPM(s)←EPM(s)∪{m}
8.    Refine(M)
9.    Refine(R)
10.   found←Node/EdgePathMappingSearch(s,M,R)
11.   if found then return true
12.   else s←RecoverState(s')
13. return false;
```

3.4 Refinement Procedure

To enumerate all possible mapping solution is time consuming, so space pruning is essential for node disjoint SHD. We devise two refinement procedures on *R* and *M* respectively, the former is based on lemma 4,5, and the latter is based on lemma 6.

LEMMA 4: In the matching process, if $v \in NM^{(2)}$, then any path with *v* as inner vertex will not $\in EPM^{(2)}$.

LEMMA 5: In the matching process, if $p \in EPM^{(2)}$, then any path passing trough the inner vertex of *p* will not $\in EPM^{(2)}$.

LEMMA 6: If $(v_i,v_j) \in NM$ ($v_i \in V_1$, $v_j \in V_2$) and assume that current state is *s* with $|M(s)|=t$ and $M(s) \subseteq NM$, then the following statements hold:

(1) $\prod r_{j'k} > 0$ ($j'=Index(j)$, $k \in Index(M(s)^{(2)})$).

(2) $\forall v' \in Adjacent(v_i)$, $\exists v \in V_2$ with $l_2(v)=l_1(v')$ and $r_{j'k} > 0$, ($j'=Index(j)$,$k=Index(v)$).

(3) The path set consisting of the paths to which $r_{j'k}$ in (1) and (2) indicates is independent.

In the above statements, index(R) gets an index in *R* for a node in G_2 and *Adjacent*(*v*) obtains the adjacent nodes set of *v*.

4. Pattern Enumeration

In this section, we focus on the following problem: given conserved structures of size *k*, *i.e.* F^k, how to efficiently enumerate all possible candidate conserved patterns C^{k+1}, *s t.* (1) *Completeness*, all conserved patterns of size *k+1*, belong to C^{k+1}; and(2) *Compactness*, for any two candidate patterns P_1, $P_2 \in C^{k+1}$, P_1 is not isomorphic to P_2.

4.1 Pattern Growth Operators

The whole pattern growth procedure can be recursively described. We begin the discussion with initial phase F^1 .

For PPI networks, each vertex in the network represent a unique protein of the specie and after data preprocessing, proteins can be labeled so that each protein is globally unique. Therefore, given F^1, let $V(F^1)=\{v_1,v_2|e(v_1,v_2)\in F^1\}$, we can define a *linear order* on $V(F^1)$, which can is a labeling function. For example, we can assign to each node a distinctive integer ID $\mu : V(F^1) \rightarrow I$, $I=\{1,2,\ldots,n\}$, where $n=|V(F^1)|$. Then each conserved edge can be uniquely identified by an *ordered node pair* $<i,j>$,*s.t* .$i \leq j$, where $i,j \in I$. Thus, we also can define a linear order $\leq_1$ on F^1 according to the lexicographical order of the corresponding *ordered node pairs*. Then in terms of order $\leq_1$, each conserved edge can be indexed by an integer $i \in [1,m]$, where $m=|F^1|$.

For each pattern P^k with size *k*, we assign a globally unique pattern ID *PID* and a bit sequence $b=b_1,\ldots,b_m$, *s.t.* $b_i=1$ *iff* the *i*-th conserved edge in F^1 occur in *P*, otherwise 0. Each pattern P^k, is generated by merging two *joinable* patterns with *k-1* size, which is called as *P's parent patterns*. Hence, we can associate to each P^k an ordered pair $<PID1, PID2>$ *s.t.* $PID1<PID2$, to indicate its two parents' pattern ID.

Similar to F^1, for each $P^k \in C^k$, we define a linear order $\leq_k$ on C^k, which is defined in terms of the lexicographical order defined on the corresponding parent ID pairs. Formally, let Z^k be the set whose elements are finite sequence of elements of C^k, the *lexicographical order of* Z^k *induced by* $\leq_k$ is denoted as $\leq_k'$, then for distinct $P_1,P_2 \in C^k$, $P_1 \leq_k P_2$ *iff* $P_1.<PID_1, PID_2> \leq_{k-1}' P_2.<PID_1, PID_2>$. Furthermore, given $\leq_k$, we assign to each $P^k \in C^k$ a globally unique pattern ID *s.t.* $P_1 \leq_k P_2 \Leftrightarrow P1.PID \leq P2.PID$.

Definition 3 (**Pattern Joinable**) Given F^k, for *any two patterns* P_1, $P_2 \in F^k$ *s.t.* $P_1 \leq_k P_2$

(1) case $P_1.PID_2=P_2.PID_1$, then P_1 *is forward joinable to* P_2
(2) case $P_1.PID_1=P_2.PID_1$, then P_1 *is backward joinable to* P_2 *in case 1*
(3) case $P_1.PID_2=P_2.PID_2$, then P_1 *is backward joinable to* P_2 *in case 2.*

Definition 4 Given F^k , for *any two patterns* P_1, $P_2 \in F^k$, we define the following binary operator on F^k:

(1) $\triangleright$ (**Forward Join Operator**)
If P_1 is *forward joinable to* P_2, $P_1 \triangleright P_2=P_1 \cup P_2$, *else* $P_1 \triangleright P_2=\emptyset$.
(2) $\triangleleft_1$ (**Backward Join Operator 1**)
If P_1 is *backward joinable to* P_2 *in case 1*, $P_1 \triangleleft_1 P_2=P_1 \cup P_2$, *else* $P_1 \triangleleft_1 P_2=\emptyset$.
(3) $\triangleleft_2$ (**Backward Join Operator 2**)
If *P1* is *backward joinable to* P_2 *in case 1*, $P_1 \triangleleft_2 P_2=P_1 \cup P_2$, *else* $P_1 \triangleleft_2 P_2=\emptyset$. Where, $P_1 \cup P_2=(V(P_1) \cup V(P_2), E(P_1) \cup E(P_2))$.

Example 3 As shown in Figure 10, given frequent pattern set F^k, using the operators defined in definition 4, we can get C^{k+1}, if all the patterns in C^{k+1} is frequent, continuously using these three operators, we get C^{k+2}. Note that, for *k*=1, P_1, P_2, P_3 in all the patterns in Figure 10 just represent a conserved node of PPI networks, respectively. For *k*>1, P_1, P_2, P_3 represent a pattern generated from F^{k-1}, respectively.

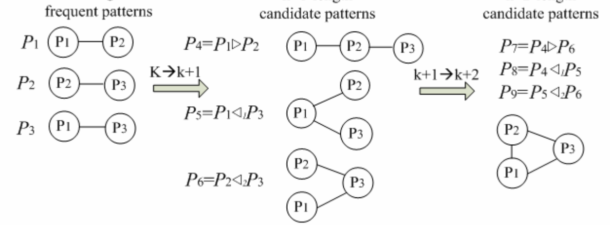


Figure 10: Pattern Growth Method

4.2 Procedure of Candidate Generation

With the above preliminaries ready, it's not difficult to sketch out the framework of candidate generation procedure, which is shown in Algorithm 4. In the algorithm, two tricky and time-consuming problems of the existing pattern growth methods can be solved in a relative lightweight method. The first is *duplication testing to avoid pattern duplication*, the second is *k-1 core testing or pattern joinable testing*.

Algorithm 4 CandidateGen(F^k)

```
Input: $F^k$ : the frequent pattern set of size $k$;
Output: $C^{k+1}$ : the candidate pattern set of size $k+1$;
Method:
1. for each $P_1 \in F^k$
2.   for each $P_2 \in F^k$ s.t. $P_1 \leq_k P_2$
3.     If $P_1$ is forward joinable to $P_2$, then $p \leftarrow P_1 \rhd P_2$
5.     If $P_1$ is backward joinable 1 to $P_2$, then $p \leftarrow P_1 \lhd_1 P_2$
6.     If $P_1$ is backward joinable 2 to P2, then $p \leftarrow P_1 \lhd_2 P_2$
7.     If $p \neq \emptyset$ and $p \notin C^{k+1}$ then
         //test if downward closure property holds for $p$
8.       flag ←true
9.       for each $p' \in C^k - F^k$
10.        If $p'$ is subpattern of $p$ then
11.          flag←false
12.          exit for
13.      if flag then $C^{k+1} \leftarrow C^{k+1} \cup \{p\}$
14. return $C^{k+1}$;
```

Pattern joinable testing in algorithm 4(line 3-6) only need to compare parent id pairs between two pattern of F^k, which avoid the time consuming procedure *to generate k-1 core of each pattern in* F^k *, to determine the joinable of these k-1 cores* .

Duplication Testing. Due to the uniqueness of labeling of PPI networks, in line 7 we only needs to determine whether $p \notin C^{k+1}$, by performing an logical 'and' computation on the bits array of each pattern. Formally, if there exists a $p' \in C^{k+1}$ *s.t.* $p.b \& p'.b = p.b$ then *p is a duplication of p'*.

Similar to many existing pattern growth methods, the infrequent pattern set C^k-F^k is used to filter the candidate pattern set in terms of the *downward closure property*. The pruning procedure is described from line 8-13 in Algorithm 4.

5. Pattern Matching As a White Box

Generally, pattern matching is not the focus of most of the existing graph mining system. While in PPI networks mining application, we need to consider the process of pattern matching process, because of the semantics of the label of the node. In Figure 10, when pattern P_1 grows into P_4 , the pattern matching of P_4 doesn't start from the scratch, but start from P_1's matched state. Out of this observation, it is rational for us to unfold the SHD procedure and integrate it into the mining procedure.

Algorithm 5 NodeDisjointSHD_WB (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 a $(l, h)$ topology minor of $G_2$ return true and return the first found node disjoint subgraph homeomorphism ($f$, $g$), otherwise return false.
Method:
// Initialize the basic data structures
1. Initial($M$, $G_1$)
2. Initial($R$, $G_1$)
3. $s$←GetState($G_1$,$S$) //initialize state as the matched state
4. $s$←NodeMappingSearch($s$,$M$,$R$) //node mapping space search
5. If not IsValid(s) return false else
6.   $s$ ←EdgePathMappingSearch($s$,$M$,$R$)
7.   $S$=$S \cup \{s\}$
8. return true;
```

However, not all the algorithm of pattern matching is convenient for the implementation of this idea. Luckily, two-level state space searching based pattern matching can be adapted to implement the idea in little difficulties. Moreover, the only thing we need to do is to modify ndSHD, as described in Algorithm 2, while the whole algorithm framework: ***Apriori-CSMiner***, as described in Algorithm 1, can retain without any change. The modified ndSHD , denoted as **NodeDisjointSHD_WB,** is desciribed in Algorithm 5. In the algorithm, we need a set S to save all the matched state of pattern matching between pattern graph G_1 and data graph G_2. S can be implemented as a hash table and each entry in the table is the pair <<*PID*, *GID*>, *MS*>, where PID is the pattern id of G_1, and *GID* is the id of the data graph G_2, and *MS* is the matching state.

6. Performance and Mining Result

We use the PPI networks of *Saccharomyces Cerevisiae*(which has 2187 nodes and 4837 edges after preprocessed, as shown in Figure1) and *Drosophila Melanogaster*(which has 404 nodes and 481 edges after preprocessed) to show the mining result of the alignment algorithm.

First, using BLAST[18], we perform protein sequences alignment between the protein from *Saccharomyces Cerevisiae,* and *Drosophila Melanogaster.* Thenwe get a similarity score table filled with the similarity value of these protein pairs. The distribution of the similarity score value is shown in Figure 11. The mean value of the similarity score is 31 (shown as the red line in Figure 11). We use $x- \bar{X} > d$ to filter out those protein pairs that is relatively less similar to each other. If we set $d=9$, then we get $x>40$, which is shown as the upper line above the mean line in Figure 11. Of course, d or α can be parameterized to tune the result of the alignment.

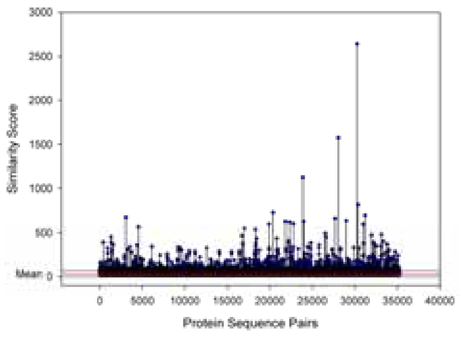
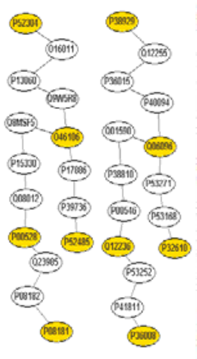
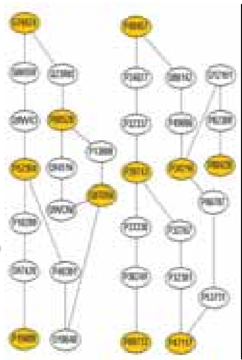
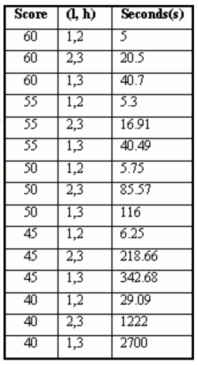


Figure 11:Protein Sequence Similarity Distribution

Let α =50, l=1, h=4, we found a conserved topological pattern as shown in Figure 12(a). Let α =40, l=2, h=3, we found a more complex conserved topological patterns as shown in Figure 12(b). We also perform a series of experiments; part of results about the performance of our system is listed in Figure 12 (c).







Score	(l, h)	Seconds(s)
60	1,2	5
60	2,3	20.5
60	1,3	40.7
55	1,2	5.3
55	2,3	16.91
55	1,3	40.49
50	1,2	5.75
50	2,3	85.57
50	1,3	116
45	1,2	6.25
45	2,3	218.66
45	1,3	342.68
40	1,2	29.09
40	2,3	1222
40	1,3	2700

(a) **(b)** **(c)**

Figure 12: Resulting Conserved Topological Structures and Performance Experiment Results

The above experiments are performed on a PC with 1.2GHZ Intel CPU and 1,128M-memory and all the code of the CSMiner system are compiled by the Visual C++.NET compiler.

7. Related Works

Recently, frequent graph pattern mining has received broad research attentions. Many efficient subgraph mining algorithms have emerged, and these algorithms can be roughly classified into two categories according to the search strategy: *Apriori based level-wise strategy* and *depth first search strategy*. AGM[1] and FSG[2] are the earliest published algorithms employing the level-wise search scheme to enumerate the recurrent subgraph patterns. Hereafter, a variety of DFS based algorithms has been proposed, including MoFa[3],gSpan[4], FFSM[5],Gaston[6].

The most related work to this paper in graph mining area is TSMiner[8]. Although TSMiner has utilized the topology minor to solve the fuzzy matching problem, the pattern matching of TSMiner is still based on subgraph isomorphism. The key idea of TSMiner is to find frequent subgraph pattern first, then test whether the found subgraph pattern can be transformed into a topology minor pattern through performing (*l*,*h*) subdivision operation on the subgraph pattern. Obviously, TSMiner can not go beyond the influence of the exact graph matching.

In the network alignment research area, Kelley *et al.*[11] first introduced an efficient computational procedure for aligning two PPI networks to identify their conserved interaction pathways, called as PathBLAST[11]. PathBlast introduce the concept of *gaps* and *mismatches* to handle the inexact matching characteristic of PPI network alignment. Sharan *et al.* [12] focus on finding conserved complexes (clique-like patterns) by comparative analysis of a pair of PPI networks. Jason *et al.* [13] propose a general and robust alignment algorithm framework: Graemin, which is scalable to find conserved function modules from *large dense* interaction networks.

8. Conclusion

In this paper, we propose an Apriori based algorithm framework CSMiner to mine conserved topological structures from multi PPI networks, so that multi PPI network alignment can be efficiently performed. We also discuss the technique to improve the performance of process.

As future work, we will carry out extensive experiments to explore the characteristics of CSMiner. Moreover, we will perform comprehensive comparison between CSMiner and TSMiner, the benchmark of inexact graph pattern mining.

9. REFERENCES

[1] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD'00*, pages 13-23, Lyon, France, Sept. 2000.

[2] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM'01*, pages 313-320, San Jose, CA,Nov. 2001.

[3]. C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *ICDM*, pages 51–58, 2002.

[4] Y. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM'02*, Maebashi, Japan,December 2002.

[5]J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*, pages 549–552, 2003.

[6]S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *KDD*, pages 647–652, 2004.

[7]J. Huan, W. Wang, J. Prins, and J.Yang. Spin: mining maximal frequent subgraphs from graph databases. In *KDD'04*, pages 581–586, Seattle,USA, Aug,2004.

[8]R. Jin, C.Wang, D. Polshakov, S. Parthasarathy, G. Agrawal: Discovering frequent topological structures from graph datasets. In *KDD'05*, pages 606-611, Chicago,USA,Aug,2005.

[9]J.Chen, W.Hsu, M.Li lee, S. Ng, NeMoFiner: Dissecting gonome-wide protein-protein interactions with meso-scale networks motifs. In *KDD'06*, pages 106-115, USA,Aug, 2006

[10] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*,298:824–827, 2002.

[11]R.B. Kelley, R. Sharan, R. Karp, T. Sittler, D. Root, B. Stockwell, and T. Ideker. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. PNAS,100(20): 11394-11399 ,2003

[12]R. Sharan, T. Ideker, B.P. Kelley, R. Shamir, and R.M. Karp. Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. In RECOMB '04: pages 282–289, USA, 2004.

[13] J. Flannick, A. Novak, B. S. Srinivasan, H. H. McAdams, and S. Batzoglou: Graemlin: General and robust alignment of multiple large interaction networks. Genome Res. 16, 1169-1181, 2006

[14] DIP:http://dip.doe-mbi.ucla.edu/

[15]H. Bunke and K. Shearer: A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters, Vol.19 (1998) \255-259.

[16] N.J. Nilsson: Principles of Artificial Intelligence. Springer-Verlag,(1982).

[17]Reinhard Diestel. Graph Theory, Springer-Verlag.(2000).

[18] Altschul, S. F., W. Gish, W. Miller, E. W. Myers, and D. J. Lipman J. Mol. Biol. 215 (1990): 403-410

[19]Neil Robertson and P.D.Seymour: Graph minors. XIII: The disjoint paths problem, Journal of Combinatorial Theory. Vol.63 (1995) 65-110.