Mining Closed Relational Graphs with Connectivity Constraints*

Xifeng Yan
Computer Science
Univ. of Illinois at Urbana-Champaign
xyan@uiuc.edu

X. Jasmine Zhou
Molecular and Computational Biology
Univ. of Southern California
xjzhou@usc.edu

Jiawei Han
Computer Science
Univ. of Illinois at Urbana-Champaign
hanj@uiuc.edu

ABSTRACT

Relational graphs are widely used in modeling large scale networks such as biological networks and social networks. In this kind of graph, connectivity becomes critical in identifying highly associated groups and clusters. In this paper, we investigate the issues of mining closed frequent graphs with connectivity constraints in massive relational graphs where each graph has around $10K$ nodes and $1M$ edges. We adopt the concept of edge connectivity and apply the results from graph theory, to speed up the mining process. Two approaches are developed to handle different mining requests: CloseCut, a pattern-growth approach, and Splat, a pattern-reduction approach. We have applied these methods in biological datasets and found the discovered patterns interesting.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications - Data Mining

General Terms: Algorithms

Keywords: graph, closed pattern, connectivity

1. INTRODUCTION

Graphs are natural representations of complicated structures and relationships among objects. Algorithms proposed in [9, 10, 22, 17, 1, 8] can find frequent subgraphs efficiently in chemical compound datasets. These algorithms have been successfully used in protein classification [8] and graph indexing [24]. There exists a specific kind of graph structure, called *relational graph*, where each node label is used only once per graph. Relational graph is widely used in modeling and analyzing massive networks, e.g., biological networks, social networks, transportation networks and the world wide web. In biological networks, nodes represent objects like genes, proteins and enzymes while edges encode the relationships such as control, reaction and correlation between these objects. In social networks, each node represents a unique entity and an edge describes a kind of relationship between entities. For instance, the Digital Bibliography & Library Project (DBLP) records multiple social networks such as co-author relations and article-reference relations.

One particular interesting pattern is frequent highly connected subgraph in large relational graphs. In social networks, this kind of pattern can help identify groups where people are strongly associated. In computational biology, highly connected subgraph could represent a set of genes within the same functional module, i.e., a set of genes participating in the same biological pathways [3, 14]. Butte et al. [3] calculates the pair-wise similarity between gene expressions to construct relevance networks in order to discover functional relationships between genes. Since a functional module shall be active under multiple relevance networks, a challenging problem is "can we discover highly connected subgraphs conserved in multiple relevance networks?"

The common problem in the above application scenario is to find not only frequent graphs, but also graphs that satisfy the connectivity constraint. Figure 1 depicts our problem setting: *how to mine frequent highly connected subgraphs in a set of massive relational graphs.* Note that the patterns we are interested in are not only groups of highly connected objects, but also the structures within these objects, which expose the relationships between the objects.

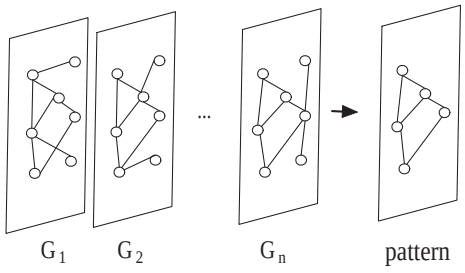

Figure 1: Mining Massive Relational Graphs

This new problem setting has three major characteristics different from the previous frequent graph mining problem defined in [9, 10, 1, 22, 17]. First, in relational graphs each node represents a distinct object. No two nodes share the same label. In biological networks, nodes often represent unique objects like genes and enzymes. Secondly, relational

*The work was supported in part by U.S. National Science Foundation NSF IIS-02-09199, Univ. of Illinois, and an IBM Faculty Award.

KDD'05, August 21–24, 2005, Chicago, Illinois, USA.

graphs may be very large. For example, gene relevance networks often have thousands of nodes and millions of edges. Thirdly, the interesting patterns should not only be frequent but also satisfy the connectivity constraint. Previous studies usually interpret a frequent graph as an object and ignore its internal properties such as connectivity.

In order to handle these new challenges, two issues have to be solved: (1) how to mine frequent graphs efficiently in large relational graphs, and (2) how to handle the connectivity constraint. Since frequent graph mining usually generates too many patterns, as observed in [2, 25, 18, 23], it is more appealing to mine closed frequent graphs only. A frequent graph is *closed* if and only if there does not exist a supergraph that has the same support. Assume graphs g and g' are frequent subgraphs and appear in the same set of relational graphs. If g is contained by g', then it is unnecessary to present graph g to users since it does not provide new information. We say g is not closed. We develop an efficient algorithm, CSPAN, to address the first problem. Unless specifically noted, the patterns discussed in the paper are *closed frequent graphs with connectivity constraints*. Since relational graphs can be represented as sets of distinct edges, we adopt the frequent itemset mining technique. However, we cannot directly cast this problem into a standard frequent itemset mining problem due to two reasons. First, the solution should assure the discovered graphs are connected. Second, we have connectivity constraints. Simply applying frequent itemset mining may immediately explode the pattern space.

Our major contribution is to tackle the connectivity constraint. We use the minimum cut criterion to measure the connectivity of a pattern and examine the issues of integrating the connectivity constraint with the closed graph mining process. As suggested by many graph theoretic approaches for data clustering [21, 13, 6], minimum cut measurement, also called edge connectivity, is good at clustering nodes based on their connectivity. The fastest deterministic minimum cut algorithm in practice has time complexity $O(|V||E| + |V|^2 log|V|)$, where $|V|$ and $|E|$ are the node set size and the edge set size of a given graph [4, 15]. To compute the edge connectivity in all subgraphs of closed graphs is equivalent to enumerating all frequent graphs and check their connectivity. Since billions of frequent graphs may exist in the datasets we are dealing with, it is impossible to finish the brute-force computation within limited time. Thus, we develop two graph theoretic approaches, CLOSECUT (a pattern-growth approach) and SPLAT (a pattern-reduction approach), to efficiently discover closed highly connected graphs while still preserving the completeness of the mining result. We apply graph condensation and decomposition techniques in the design of CLOSECUT and SPLAT to improve the performance. Both of them can reduce the size of candidate graphs in terms of nodes and edges. CLOSECUT and SPLAT are targeted to handle different mining requests. Their pros and cons will be illustrated through our experiments.

The contribution of this study is not only providing an affordable solution to mine highly connected graphs in multiple relational graphs, but also the demonstration of how frequent graph mining technology may help uncover interesting patterns in scientific fields like biology. We have applied CLOSECUT and SPLAT in real biological datasets and found the discovered patterns very promising. Although the patterns mined by our methods need post-processing to extract other variant dense graphs such as cliques, the most frequent patterns mined by our methods exhibit strong biological meanings and are directly usable.

The remaining of the paper is organized as follows. Section 2 gives the problem definition. The theorems leading to the efficient design of CLOSECUT and SPLAT are introduced in Section 3, followed by a detailed explanation of CLOSECUT in Section 4 and SPLAT in Section 5. We report our performance result in Section 6. Related work is discussed in Section 7, and Section 8 concludes our study.

2. FORMULATION

A relational graph set consists of undirected simple graphs, $\{G_i = (V, E_i)\}$, $i = 1, \ldots, n$, $E_i \subseteq V \times V$, where a common vertex set V is shared by the graphs in the set. In relational graphs, there is neither loop nor multiple edges. Unless otherwise specified, all the graph patterns discussed in this paper are *undirected connected* relational graphs.

A relational graph $G = (V, E)$ is a subgraph of $G' = (V, E')$ if and only if $E \subseteq E'$, denoted by $G \subseteq G'$ (G' is a supergraph of G). Graphs G and G' are isomorphic if and only if $E = E'$. The testing of subgraph isomorphism between two relational graphs is easy because labels have to match exactly and each label is used only once in a graph. We denote the vertex set of a graph G by $V(G)$ and the edge set by $E(G)$.

In order to determine whether one set contains another set, one can first sort the two sets and then compare them in linear time. Hence, the complexity of subgraph isomorphism testing of relational graphs is $O(|E|log|E|)$, where $|E|$ is the size of the larger graph in two graphs. We define union, intersection and difference operators for relational graphs.

DEFINITION 1 (UNION, INTERSECTION, DIFFERENCE). *Given two relational graphs, $G = (V, E)$ and $G' = (V, E')$, the union of G and G', written $G \cup G'$, is $(V, E \cup E')$. The intersection of G and G', written $G \cap G'$, is $(V, E \cap E')$. The difference of G and G', written $G - G'$, is $(V, E \setminus E')$.*

DEFINITION 2 (CONSTRAINT). *A constraint is a function, $C : \{G\} \rightarrow \{0, 1\}$, which maps a graph G to a Boolean value. A graph G satisfies constraint C if $C(G) = 1$.*

Given a relational graph dataset, $\mathbb{D} = \{G_1, G_2, \ldots, G_n\}$, $support(g)$ is the number of relational graphs (in $\mathbb{D}$) where g is a subgraph. Let $F = \{g |\ support(g) \geq min\_sup\ and\ C(g) = 1\}$, where min_sup is an non-negative integer and C is a constraint. F is called a *frequent graph set with constraint* C. Often the closed set of F is more interesting: $\{g | g \in F, and\ \nexists\, g' \in F\ s.t., g \subset g'\ and\ support(g) = support(g')\}$. The reason of mining closed graphs is to avoid the exponential number of patterns in large graph datasets. In our study, we shall concentrate on implementing the *connectivity* constraint and mining closed frequent graphs with connectivity constraints. As observed in our experiments, the number of such patterns is much less than closed frequent graphs.

DEFINITION 3 (DEGREE). *The degree of a vertex v is the number of edges that connect v, written as $degree(v)$. The average degree of a graph G is the average of $degree(v)$ for all $v \in V(G)$. The minimum degree of a graph G is the minimum of $degree(v)$ for all $v \in V(G)$, written as $\sigma(G)$.*

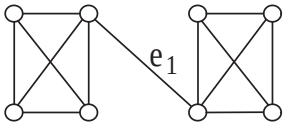

Figure 2: Average Degree:3.25, Minimum Degree:3

Although average degree and minimum degree display some level of connectivity in a graph, they cannot guarantee the graph is connected in a balanced way. Figure 2 shows an example that some part of a graph may be loosely connected even if its average degree and minimum degree are high. The removal of edge e_1 will make the whole graph fall apart. One may enforce the following downward closure constraint: a graph is highly connected if and only if each of its connected subgraphs is highly connected. However, some global tightly connected graphs may not be locally connected well. It is a bit too strict to have this downward closure constraint. Thus, we adopt the concept of edge connectivity, a well-known connectivity definition in graph theory.

Definition 4 (Edge Connectivity). *Given a graph G, an edge cut is a set of edges E_c such that $E(G) - E_c$ is disconnected. A minimum cut is the smallest set in all edge cuts. The edge connectivity of G, written $\kappa(G)$, is the size of a minimum cut.*

A cut E_c separates $V(G)$ into two vertex sets, V and $\widetilde{V}$, such that all the edges in E_c are only edges between V and $\widetilde{V}$, where $V \cap \widetilde{V} = \varnothing$ and $V \cup \widetilde{V} = V(G)$. E_c is also written as $V \rightarrow \widetilde{V}$ to show that edges in E_c connect V and $\widetilde{V}$. For example, Set $\{e_1\}$ is a minimum cut of the graph shown in Figure 2, which separates the graph into two equal-sized components. Edge connectivity/minimum cut is popularly used to cluster objects in a graph [21, 13, 6].

In this paper, we investigate the issues of *mining all closed frequent graphs with edge connectivity (minimum cut size) at least K*, where K is a natural number.

3. PROPERTIES OF CONNECTIVITY

We first examine several graph theoretic concepts about edge connectivity.

Claim 1 (No Downward Closure Property). *Given two graphs G and G', $G \subset G'$ and $\kappa(G) \leq \kappa(G')$ do not imply each other.*

Claim 1 says that the high connectivity of a graph does not imply the high connectivity of its supergraph, and vice versa. There is no downward closure property for edge connectivity. However, two specific kinds of graphs, clique and tree, have the downward closure property. If a graph is a clique or a tree, then all its induced connected subgraphs are cliques or trees. As one can see, cliques and trees are either too strict or too sparse. They may not catch some interesting graph patterns. In fact, the lack of downward closure in edge connectivity brings more flexibility on the kind of patterns we may find.

Wu and Leahy [21] examined the properties of edge connectivity. We derive two corollaries from Theorem 4 in [21] and give self-contained proofs. We are not aware of any previous work which applies these results in mining highly connected patterns through multiple graphs, which will be addressed in the next section.

Corollary 1 (Condensation). *Let G be a subgraph of a graph G', and G^* be the graph formed from G' with all vertices in G condensed into a single vertex. If $\kappa(G) > \kappa(G')$, then $\kappa(G^*) = \kappa(G')$.*

Proof. Let $V_m \rightarrow \widetilde{V}_m$ be the minimum cut of G'. Since $\kappa(G) > \kappa(G')$, then $V(G)$ must be a subset of V_m or a subset of $\widetilde{V}_m$; otherwise, $\kappa(G') \geq \kappa(G)$. In either case, $V_m \rightarrow \widetilde{V}_m$ is an edge cut of G^*. Therefore, $\kappa(G^*) \leq \kappa(G')$. Let $V_m^* \rightarrow \widetilde{V}_m^*$ be the minimum edge cut of G^*. $V_m^* \rightarrow \widetilde{V}_m^*$ is an edge cut of G'. Thus, $\kappa(G^*) \geq \kappa(G')$. Hence, $\kappa(G^*) = \kappa(G')$. ∎

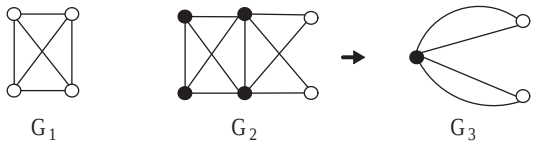

Figure 3: Condensation

Example 1. *Figure 3 shows an example of condensation. The edge connectivity of graphs G_1, G_2, and G_3 are 3, 2, and 2 respectively. We have $\kappa(G_1) > \kappa(G_2)$ and $G_1 \subset G_2$. It is safe to condense all the vertices of G_1 into a single vertex to form a new graph G_3. G_3 has the same edge connectivity as G_2.*

Corollary 1 shows that we may reduce the cost of calculating edge connectivity if the connectivity of its subgraph is known. This property is useful for the design of our pattern-growth approach, CloseCut. For example, suppose we have already discovered a highly connected graph pattern g and extend it to a new candidate pattern g'. We can form a new graph g^* by condensing all the vertices of g into a single vertex in g'. If we want to know the edge connectivity of g', we only need to check the connectivity of g^*. Since g^* is smaller than g', the computation cost of edge connectivity will be reduced. Note that g^* is not a simple graph any more. It may have multiple edges between two vertices. Corollary 1, as well as Corollary 2, is also valid for graphs with multiple edges.

Corollary 2 (Exclusion). *Let G be a subgraph of a graph G' and E_c be an edge cut of G' such that $|E_c| < K$. If $\kappa(G) \geq K$, then $E_c \bigcap E(G) = \varnothing$.*

Proof. Let $V \rightarrow \widetilde{V}$ be the edge cut E_c in G'. Since $\kappa(G) > |E_c|$, then $V(G)$ must be a subset of V or a subset of $\widetilde{V}$; otherwise, E_c become a superset of an edge cut of G, hence $|E_c| \geq \kappa(G)$. In either case, $\forall e \in E_c, e \notin E(G)$. Hence, $E_c \bigcap E(G) = \varnothing$. ∎

Example 2. *Figure 4 shows an example of exclusion. G_2 is a subgraph of G_1. G_1 has an edge cut $\{e_1, e_2\}$. Assume we want to find graph patterns with edge connectivity at least 3. According to Corollary 2, if a subgraph of G_1 has edge connectivity at least 3, it will not have edges e_1 and e_2. So the deletion of e_1 in G_2 will not lose any pattern.*

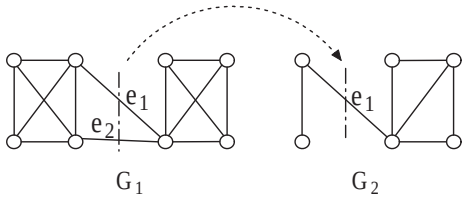

Figure 4: Exclusion

Corollary 2 shows if a graph has an edge cut whose size is less than K, then none of its subgraphs with edge connectivity at least K will contain the edges in this cut.

If the edge connectivity of a closed frequent graph is less than K, we have to find its subgraphs that have edge connectivity at least K. Actually, each of them is a subgraph defined in the maximum K-*decomposition* (Definition 5), a variant of K-partition introduced in [21]. K-decomposition breaks a graph into non-overlapping subgraphs such that their connectivity is at least K. Using Corollary 3.2 in [21], we can prove that the maximum K-decomposition is unique.

DEFINITION 5 (K-DECOMPOSITION). *The K- decomposition of an undirected graph G is a set of subgraphs $\{g_i\}$, $g_i \subseteq G$, s.t. $\kappa(g_i) \geq K$ and $g_i \cap g_j = \varnothing$. The maximum K-decomposition is a K-decomposition that maximizes $\sum |E(g_i)|$.*

We can construct a K-decomposition in a divide-and-conquer manner: (1) select an edge cut whose size is less than K in graph G (if there is such a cut); (2) decompose G into two subgraphs by removing the cut edges; (3) recursively call Steps 1 and 2 on every decomposed subgraph until its edge connectivity is at least K or it becomes a single vertex. The K-decomposition obtained in this way must be the maximum K-decomposition and the only maximum K-decomposition that G has.

Using K-decomposition to break a graph, we will not miss any subgraph whose edge connectivity is at least K. Furthermore, the divide-and-conquer property of K- decomposition makes the mining affordable. We can further apply Corollary 1 to condense a graph before we perform the decomposition. For example, if a graph G has a subgraph whose connectivity is at least K, we can condense all the vertices of this subgraph into a single vertex to form a new graph G^*. We then decompose G^* instead of G to obtain the highly connected subgraphs in G. Since G^* is smaller than G, the decomposition performance will be improved.

4. CLOSECUT: A PATTERN GROWTH APPROACH

In this section, we formulate our first algorithm, CLOSECUT, for mining closed frequent graphs with connectivity constraint. CLOSECUT adopts a pattern-growth approach: It first finds a small frequent candidate graph and decomposes it to extract the subgraphs satisfying the connectivity constraint. After that, CLOSECUT extends the candidate graph by adding new edges and repeats the above operations. Before we discuss how to integrate the connectivity constraint with the mining process, we first examine how to mine closed frequent graphs from relational graph datasets.

4.1 Closed Frequent Graph Mining

Different from general labeled graphs, each node in relational graphs has a unique label per graph. Because of this special property, we can treat relational graphs as sets of edges (v_i, v_j) and use the closed frequent itemset mining technique instead of general graph mining algorithms.

Algorithm 1 CSPAN(g, D, *min_sup*, S)

Input: A graph g, a graph dataset D, a minimum support threshold *min_sup*.
Output: The closed frequent graph set S.

1: **if** $\exists g' \in S$, $g \subset g'$ and $support(g) = support(g')$ **then return**;
2: extend g to g' as much as possible s.t. $support(g) = support(g')$;
3: insert g' to S;
4: scan D once, find every edge e s.t. $g' \cup \{e\}$ is frequent;
5: **for each** frequent $g' \cup \{e\}$ **do**
6: CSPAN($g' \cup \{e\}$, D, *min_sup*, S);
7: **return**;

Algorithm 1 (CSPAN) illustrates the framework of our closed frequent graph mining engine. It adopts the pattern-growth approach. A new graph is first extended from a small frequent graph with new edges added in. Then the frequency of the new graph is checked. In each iteration, CSPAN extends a newly discovered frequent graph as much as possible until it finds the largest supergraph with the same support. Using this technique, many iterations that do not generate closed graphs can be skipped. For example. Suppose there is a set of frequent graphs g_1, g_2, ..., g_n, where graph g_i is formed from g_{i-1} by adding one new edge ($1 < i \leq n$). If graphs $g_1, g_2, \ldots$, and g_n have the same support, one could skip the search space between g_1 and g_n. This strategy lowers down the computation cost. Consequently, the graph found in Line 2 is a closed frequent graph. Lines 4-6 discover the remaining supergraphs that have lower support. According to the following lemma, there is one and only one closed graph that can be extended from each frequent graph in Line 2, Algorithm 1.

LEMMA 1 (UNIQUENESS). *Given a relational graph G, there is one and only one closed relational graph G' such that $G \subseteq G'$ and $support(G) = support(G')$.*

Proof. Assume to the contrary that there is another closed graph G'' s.t. $G \subset G''$ and $support(G) = support(G'')$. Let G^* be the graph formed by $G' \cup G''$. G^* is a connected graph since G' and G'' share a common subgraph G. Therefore, $G'' \subset G^*$ and $support(G'') = support(G^*)$, contradicting our assumption. ∎

Lemma 1 does not hold for general labeled graphs. Because multiple subgraph isomorphisms may exist between general graphs, we cannot claim in the proof that G^* is a connected graph any more.

4.2 Edge Connectivity Constraint

In the next section, we examine an issue raised by the edge connectivity constraint when we attempt to integrate this constraint with Algorithm 1.

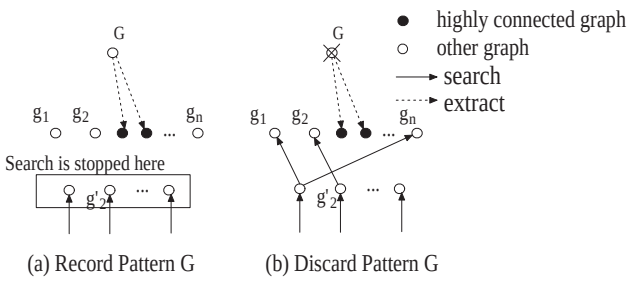

Figure 5: Search Space

Suppose we find a closed frequent graph G that does not satisfy the connectivity constraint during mining, should we record it or not? The answer depends on what we want to optimize: space or time. Figure 5 shows an example of this issue. Assume G has an exponential number of subgraphs $g_1, g_2, \ldots, g_n$ such that $g_i \not\subseteq g_j$ for any i and j. Suppose G and $\{g_i\}$ have the same support and some graphs in $\{g_i\}$ satisfy the connectivity constraint. After we extract the highly connected graphs from G, we can either record G or discard G. If we discard G, a potential problem arises: one probably has to generate $g_1, g_2, \ldots, g_n$ individually in order to determine whether they satisfy the constraint. Since G is not recorded, the search space below g_i cannot be skipped. For example, assume g_2 is expanded from g_2' and they have the same support. If we retain G in the memory, as shown in Figure 5(a), we can immediately stop searching any supergraph of g_2' since g_2' is a subgraph of G and they have the same support. However, if G is discarded, we have to grow g_2 from g_2' because there is no clue that the supergraphs of g_2' have already been checked. Considering a lot of graphs have the similar condition as g_2, it is inefficient to generate them separately. It takes more time (by orders of magnitude) to complete the mining. Therefore, we decide to record closed graphs that do not satisfy the connectivity constraint. Although it incurs additional space cost, we believe the shorter computation time is the key issue in our applications.

4.3 The Framework of CloseCut

We build CloseCut based on Algorithm 1, incorporating the connectivity constraint. When it extends a frequent graph, CloseCut attempts to remove the frequent edges that will not be part of highly connected graphs.

Algorithm 2 sketches the framework of CloseCut, which consists of four steps: (Step 1) find the closed graph of a newly discovered graph (Lines 2-3); (Step 2) condense and decompose graphs for highly connected subgraphs (Lines 5-7); (Step 3) remove frequent edges of unpromising vertices (Lines 8-9); and (Step 4) recursively search new graphs (Lines 10-11). According to the discussion in the previous section, we record the intermediate mining results in a set (set C in Algorithm 2). These intermediate results are used to avoid extending the same frequent graph twice. Duplicate extensions are blocked by Line 1 in Algorithm 2. Corollary 1 is used to accelerate the computation of edge connectivity in the second step, shown in Line 5. The following sections will introduce how to apply the minimum degree constraint in the third step and how to decompose graphs efficiently in the second step.

Algorithm 2 CloseCut(g, D, *min_sup*, K, C, S)

Input: A graph g, a graph dataset D, a minimum support threshold *min_sup*, connectivity constraint K, the previously discovered frequent graph set C.
Output: The result set S.

1: **if** $\exists g' \in C$, $g \subset g'$ and $support(g) = support(g')$ **then return**;
2: extend g to g' as much as possible s.t. $support(g) = support(g')$;
3: insert g' to C;
4: $g^* = g'$;
5: **if** $\exists g_o \in S$, g' is extended from g_o **then** condense the vertices of g_o into a single vertex in g^*;
6: decompose(g^*, K, S);
7: scan D once, find frequent edge set X, s.t. $\forall e \in X$ graph $g' \cup \{e\}$ is frequent;
8: **for each** vertex v in g', $\widehat{deg}(v) \leq K$ **do**
9: remove all edges of v in X;
10: **for each** frequent graph $g' \cup \{e\}$, $e \in X$ **do**
11: CloseCut($g' \cup \{e\}$, D, *min_sup*, K, C, S);
12: **return**;

4.4 Minimum Degree Constraint

For any graph, its edge connectivity is less than or equal to its minimum degree. This property shows that if a graph satisfies the edge connectivity constraint, it must satisfy the minimum degree constraint first. We integrate the minimum degree constraint with the mining process.

THEOREM 1. *[20]* $\kappa(G) \leq \sigma(G)$.

Suppose a newly discovered frequent graph g has its minimum degree less than K. We cannot stop searching its supergraphs since they may satisfy the constraint. However, we can check the maximum degree of vertex v ($v \in V(g)$) in the largest possible frequent supergraph that may be extended from g. If the degree of v is less than K, we can safely exclude v from g.

DEFINITION 6 (SHADOW GRAPH). *Let G be a frequent graph and X be a set of edges which can be added to G such that $G \cup \{e\}$ ($e \in X$) is connected and frequent. Graph $G \cup X$ is called the shadow graph of G, written as $\widehat{G}$. The degree of v ($v \in V(G)$) in the shadow graph of G is written $\widehat{deg}(v)$.*

For any vertex v in $V(g)$, $\widehat{deg}(v)$ is the maximum number of edges that v may have for any potential frequent supergraph of g. $\widehat{deg}(v)$ monotonically decreases when g is extended. If $\widehat{deg}(v)$ is less than K, v can be excluded from g when we extend g. In our implementation of CloseCut, we do not delete v from g. Instead, we remove all the edges of v in the frequent edge set X. By doing so, v loses the ability to have new edges. Since these frequent edges do not show up in the following extensions of g, we reduce the computation time.

4.5 Minimum Cut Decomposition

Once a frequent graph is generated in Line 2, Algorithm 2, we will extract highly connected subgraphs in Line 6, Algorithm 2. Algorithm 3 outlines the extraction procedure.

Line 1 gives a termination condition to avoid duplicate decomposition. Line 3 checks whether the discovered frequent graph satisfies the constraint. If it does, we put it in the result set. Otherwise, we recursively decompose it in Lines 6-9 until it meets the stop condition.

Algorithm 3 decompose(g, K, S)

Input: A graph g and connectivity threshold K.
Output: The result set S.

1: **if** (stopDecompose(g)) **then**
2: **return**;
3: **if** ($\kappa(g) \geq K$) **then**
4: insert g into S;
5: **return**;
6: **while** there exists a cut in g whose size is less than K
7: break g into two parts g_1 and g_2;
8: decompose(g_1, K, S);
9: decompose(g_2, K, S);
10: **return**;

When we break one graph into two halves, one or both of them may be decomposed somewhere else. Thus, a termination condition (Line 1, Algorithm 3) is set up so that we do not decompose the same graph repeatedly. For example, assume two frequent graphs g_1 and g_2 share a common subgraph g_o. If we perform a brute force decomposition on both of them, we may decompose g_o twice. Therefore, before g_0 is decomposed, we have to check whether this graph was or would be decomposed elsewhere. A naive solution is to maintain a database of processed frequent graphs. The database is checked to verify whether g_o has been decomposed before. Unfortunately, this approach is costly provided that there are lots of frequent graphs in the dataset. We propose a better solution to serve as a termination condition.

THEOREM 2 (TERMINATION CONDITION). *Let g_o be a subgraph of g. If $support(g_o) \neq support(g)$, one can stop decomposing g_o.*

Proof. If the support of g_o is different from g, according to Lemma 1, there must be another closed graph containing g_o. Thus, g_o can be decomposed when we process that graph. ∎

Theorem 2 forms a termination condition of decomposition (Line 1, Algorithm 3). According to Lemma 1, given a relational graph g, there is one and only one closed graph which is a supergraph of g and has the same support. Therefore, *any graph will be decomposed at most once* in CLOSECUT.

5. SPLAT: A PATTERN REDUCTION APPROACH

CLOSECUT extends a candidate graph by inserting new edges until the candidate graph is not frequent any more. Inspired by recent work on row enumeration-based approaches [12] and intersection methods [11], we propose a pattern-reduction approach, SPLAT. Instead of enumerating graphs from small ones to large ones, SPLAT directly intersects relational graphs and decomposes them to obtain highly connected graphs.

Let pattern g be a highly connected graph in relational graphs $G_{i_1}, G_{i_2}, \ldots$, and G_{i_l} ($i_1 < i_2 < \ldots < i_l$). In order to mine patterns in a larger set $\{G_{i_1}, G_{i_2}, \ldots, G_{i_l}, G_{i_{l+1}}\}$, SPLAT intersects g with graph $G_{i_{l+1}}$. Let $g' = g \cap G_{i_{l+1}}$. The intersection will remove some edges in g that do not exist in $G_{i_{l+1}}$. Thus, the connectivity of the new graph g' may not satisfy the constraint anymore. If so, we need to decompose g' into smaller highly connected subgraphs. We progressively reduce the size of candidate graphs by intersection and decomposition. Finally, it may become zero. We call this approach a pattern-reduction approach.

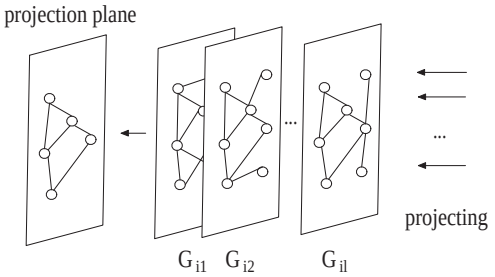

Figure 6: Splat

Figure 6 depicts the concept of SPLAT. Imagine there is a parallel light casting from the right perpendicularly to relational graphs and each edge blocks some light. The frequent edges will be very dark on the projection plane. Since we are only concerned about highly connected graphs, frequent edges with low connectivity will be removed from the plane. By inserting and deleting different relational graphs between the light and the projection plane, SPLAT is able to discover all the graph patterns satisfying the connectivity constraint. When the size of patterns on the projection plane is smaller than K, SPLAT stops intersecting it with new relational graphs since it will not produce new patterns.

Algorithm 4 SPLAT(g, D, l, *min_sup*, K, S)

Input: A graph g, a graph dataset D, an index l, a minimum support threshold *min_sup*, connectivity constraint K.
Output: The result set S.

1: check whether a discovered graph g' exists s.t. $g = g'$;
2: **if** such pattern exists **then return**;
3. **if** $support(g) \geq min\_sup$ **then** insert g into S;
4: **for each** $G_m \in D, l < m \leq n$ **do**
5: $g' = g \cap G_m$;
6: K-decompose g', put highly connected subgraphs in Q;
7: **for each** graph $q \in Q$ **do**
8: SPLAT(q, D, m, *min_sup*, K, S);
9: **return**;

Algorithm 4 describes the framework of SPLAT. Line 1 checks whether a discovered graph g exists in the result set. If g exists, SPLAT need not work on it since g will not generate new closed highly connected graphs, which can be proved using a similar framework in [12]. Lines 4-8 intersect the graph with the m-th relational graph and perform K-decomposition on it. For each newly discovered graph, we repeat the above procedure until we finish all the relational graphs or there is no new frequent highly connected graphs.

When the edge connectivity is higher, SPLAT will perform better. In that case, SPLAT shrinks the candidate graphs quickly by removing lots of low cut edges. However, the performance of SPLAT may deteriorate when the number of relational graphs increases because it has to enumerate the combination of relational graphs.

6. EXPERIMENTAL RESULTS

We conducted a comprehensive performance study on both synthetic and real datasets. The synthetic data is controlled by a set of parameters that allow us to test the performance under different conditions. The real dataset is obtained from microarray experiments. Through the experiments, we illustrate the pros and cons of CLOSECUT and SPLAT according to different mining requests.

All the experiments are done on a 2.5GHZ Intel Xeon server with 3GB main memory, running RedHat 9.0. Both CLOSECUT and SPLAT are implemented in C++ with STL library support and compiled by g++ with -O3 optimization.

6.1 Synthetic Dataset

Notation	Parameter
N	Number of relational graphs
O	Number of objects
S	Number of seed graphs
I	Maximum size of seed graphs (in vertices)
T	Average number of seed graphs
D	Average density in seed graphs
d	Average density of noise edges

Table 1: Parameters of Synthetic Data

The synthetic datasets have a set of parameters for users to specify: the number of relational graphs (N), the number of objects (O), the number of seed graphs (S), the average size of seed graphs (I), the average number of seed graphs in relational graphs (T), the average density of seed graphs (D), and the average density of noise edges in relational graphs (d). *Density* is the average degree divided by the number of vertices. Table 1 summarizes these parameters.

Synthetic data is generated as follows. First, we generate a set of seed graphs randomly. Seed graphs are used later to form the relational graphs. The total number of seed graphs is S. Their size (the number of vertices) is randomly selected between 1 and I. Let V be the number of vertices in a seed graph. We randomly label its vertices and assign $kV/2$ edges to it. The seed graph does not necessarily have exactly k edges for each vertex. Variable k is a gaussian random variable with mean D. Next, we generate the relational graphs. The numbers of objects in relational graphs are the same, specified by O. We randomly select seed graphs and embed them in the relational graphs. The number of seed graphs per relational graph is determined by a normal distribution with mean T. Finally, we randomly assign a set of noise edges to each relational graph. The number of noise edges per graph also follows a normal distribution with mean $d \times O$. When we increase the average number of seed graphs per graph, these seed graphs will be merged together to form larger irregular frequent graphs.

For a dataset which has 30 relational graphs of 10,000 distinct objects, 1,000 seed graphs (each seed graph has at most 40 vertices and an average density 0.6), 500 seed graphs per relational graph, and 100 noise edges per object ($0.005 \times 10,000 \times 2$), we represent it as N30O10kS1kT500I40 D0.6d0.005. In the following tests, we will change some major parameters including minimum support, the number of seed graphs, and average density to show the performance of CLOSECUT and SPLAT.

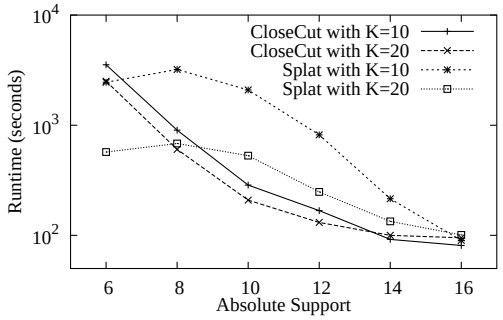

Figure 7: Runtime vs. Support

Figure 7 shows the runtime of CLOSECUT and SPLAT for dataset N30O10kS1kT500I40D0.6d0.005 with varied minimum supports. The settings of N and O are typical: a small set of graphs with large number of nodes. Each graph in this dataset has highly connected seed graphs and these seed graphs overlap with each other. The average vertex number of seed graphs is 20 in this dataset. As shown in the figure, CLOSECUT and SPLAT have the similar performance when the support is very high. The high support threshold filters out lots of infrequent edges and noise edges. Thus, both algorithms complete very fast. When the support is lowered down, CLOSECUT outperforms SPLAT because SPLAT has to enumerate lots of infrequent highly connected subgraphs, which will eventually be discarded. However, when the support is very low, the situation is reversed. CLOSECUT cannot prune effectively using the minimum degree constraint. On the contrary, SPLAT can use the connectivity constraint to remove many frequent, but low minimum cut edges. Therefore, SPLAT outperforms CLOSECUT, although both of them take a long time to finish. This explanation is also justified by the fact that the the runtime difference of $K = 10$ and $K = 20$ in SPLAT is greater than that in CLOSECUT. That means SPLAT takes more advantage of the connectivity constraint than CLOSECUT.

Having verified the runtime of CLOSECUT and SPLAT over varied supports, we then check their scalability over the number of seeds. The increment of this parameter, together with the average number of seeds per graph, will add more highly connected graphs into the dataset, thus making the mining more challenging. We increase the seed graph number from 500 to $1,500$. Figure 8 shows the runtime of these two algorithms for dataset N30O10kI40D0.6d0.005. The support threshold is 10. As shown in the figure, CLOSECUT and SPLAT are scalable to the number of seed graphs.

Figure 9 shows the scalability of these two algorithms with different density settings. When seed graphs become denser and larger in terms of edges, the number of patterns will increase and more graphs will satisfy the connectivity constraint. Both CLOSECUT and SPLAT scale well in this case.

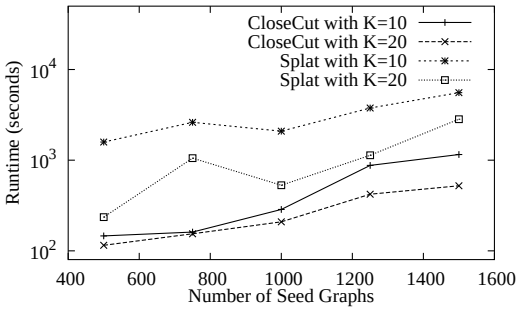

Figure 8: Runtime vs. Number of Seed Graphs

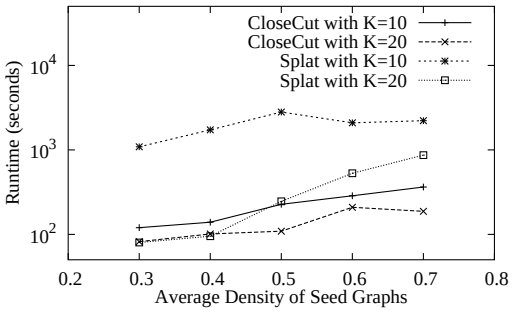

Figure 9: Runtime vs. Density

6.2 Real Dataset

The real data consists of 32 microarray expression sets measuring yeast genome-wide expression profiles under different types of perturbations, e.g., cell cycle, amino acid starvation, heat shock, and osmotic pressure. Each dataset includes the expression values of 6661 yeast genes over multiple conditions. Since gene expression values generated by different platforms are not comparable, we cannot calculate their correlation across all the datasets directly. Instead, we model each dataset as a relational graph, where nodes represent genes, and we connect two genes with an edge if they have high correlation in their expression profiles [5, 16]. On average, each graph have around 600,000 edges. In general, genes with correlated expression profiles are likely to be functionally related. We applied CloseCut and Splat in this microarray dataset and mined recurrent graphs with different connectivity constraints.

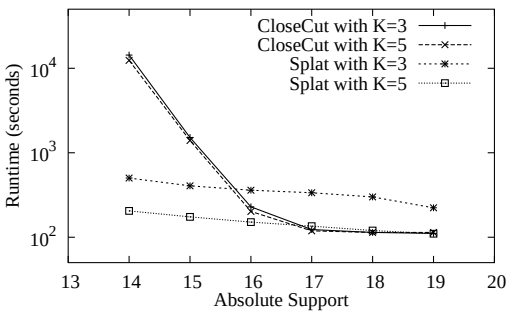

Figure 10: Runtime vs. Support

In Figure 10, we plot the runtime of CloseCut and Splat in this microarray dataset. For this dataset, Splat outperforms CloseCut when the absolute support is below 16. However, CloseCut has better performance when the minimum support is high and the edge connectivity is low. For example, when the support threshold is set at 17 and the minimum connectivity is 2, Splat needs 2,748 seconds to finish the mining. If the connectivity is lowered down to 1, Splat cannot complete the task in hours. In both cases, CloseCut only takes less than 300 seconds. Compared with its performance in synthetic datasets, CloseCut does not achieve a similar speedup when the minimum cut threshold is increased. We found that this dataset has lots of large frequent tree patterns and many vertices in the center of these patterns have high degrees. Therefore, CloseCut cannot remove these nodes quickly using the minimum degree constraint.

In summary, CloseCut and Splat have their own strength for different problem settings. CloseCut runs faster when the support is high and the connectivity constraint is low. On the contrary, Splat has better performance when the support is low and the connectivity is high.

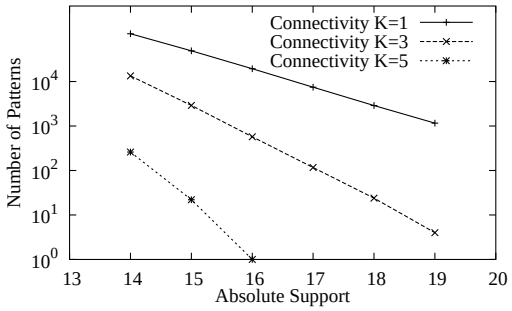

Figure 11: Number of Highly Connected Patterns

Figure 11 shows the number of highly connected graphs mined under different connectivity constraints. As shown in the figure, the number of patterns is reduced significantly when we change the connectivity threshold from 1 to 5. The result set produced from our algorithm is not only smaller than the set of frequent graphs, but also smaller than the set of closed frequent graphs.

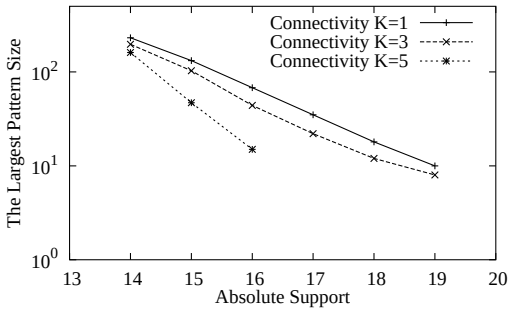

Figure 12: Size of the Largest Patterns

Figure 12 shows the size of the largest patterns discovered for different connectivity thresholds. We have small-size patterns when the connectivity constraint is enhanced.

Figures 13-16 depict examples of the most frequent graphs we discovered with edge connectivity equal to 3. The support of these patterns is all above 19. These patterns have strong biological meanings as verified by biologists.

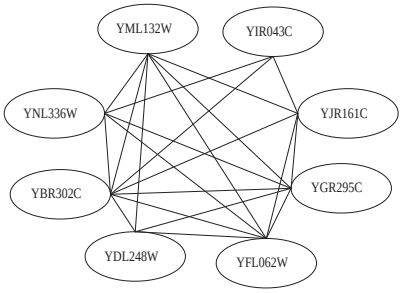

Figure 13: Genes Related with Subtelomerically Encoded Proteins

Except that we have no knowledge about gene YIR043C in the pattern shown in Figure 13, all of the rest seven genes belong to a family of conserved, often subtelomerically encoded proteins. These seven genes are located closely to each other in the chromosome. Below in the parenthesis are the common names of these genes, and one can see that they differ only in the last number: YML132W (COS3), YIR043C (unknown), YJR161C (COS5), YGR295C (COS6), YFL062W (COS4), YDL248W (COS7), YBR302C (COS2), and YNL336W (COS1).

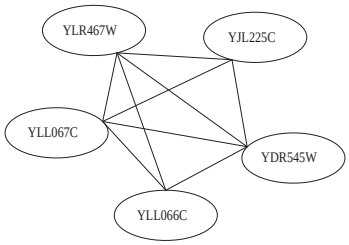

Figure 14: Genes Having Helicase Activity

All the five genes shown in Figure 14, YLR467W, YJL225C, YDR545W, YLL066C, and YLL067C, have helicase activity. However, the exact biological pathways, in which these genes are involved, are unknown so far.

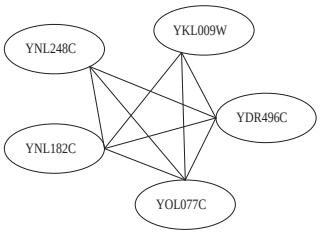

Figure 15: Genes Involved in Ribosomal Biogenesis

The five genes in Figure 15 are mainly involved in ribosomal biogenesis. Genes YKL009W, YNL182C, and YOL077C are involved in ribosomal large subunit assembly and maintenance; YNL248C is involved in transcription of ribosomal DNA; and YDR496C has unknown function, but is predicted to be involved in ribosomal RNA processing in our own study. All genes in Figure 16 are involved in rRNA processing, except that YPL146C has unknown function.

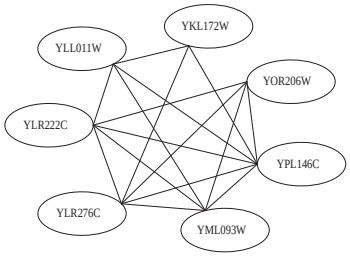

Figure 16: Genes Involved in rRNA Processing

7. RELATED WORK

A general review on the recent progress of graph-based data mining is given by Washio and Motoda [19]. The Apriori property is applied to mine frequent graphs: Inokuchi et al. [9], Kuramochi and Karypis [10], and Vanetik et al. [17]. Borgelt and Berthold [1], Yan and Han [23], and Huan et al. [8] apply the pattern-growth approach to directly mine frequent subgraphs. Holder et al. [7] adopt the principle of minimum description length for mining approximate frequent graphs. In this paper, we introduce a different problem scenario where connectivity is used together with frequency to identify highly connected recurrent structures. Furthermore, the relational graph we studied is very special in comparison with the general graph proposed by previous work. Each node in a relational graph has a distinct label. This special property leads to a different but more efficient design in mining large graph patterns.

The minimum cut criterion in finding highly connected components has been introduced in various fields. It is presented by Wu and Leahy [21] as an optimal graph theoretic approach for data clustering in the image segmentation problem. Their approach is further modified by Shi and Malik [13] using a normalized minimum cut measurement. Flake et al. discuss how to apply a framework of maximum flow/minimum cut to identify members of web communities efficiently [6]. These studies focus on clustering objects in one graph, instead of a set of graphs. It is unknown whether these techniques are still valid or applicable in the context of mining multiple graphs. For example, in the design of pattern-growth approach, it is not clear how to extend a frequent graph with concerns on the connectivity constraint and when to stop decomposing a candidate graph. So far, we are not aware of any previous work on these issues.

8. CONCLUSIONS

In this paper, we introduced a new graph mining problem: finding closed frequent graphs with connectivity constraints in relational graphs. We adopted the concept of edge connectivity and applied graph theoretic results, graph condensation and decomposition, in our algorithm design. Two approaches were developed to meet different mining demands: CloseCut, a pattern-growth approach, and Splat, a pattern-reduction approach. CloseCut has better performance on patterns with high support and low connectivity.

On the contrary, SPLAT can remove frequent graphs with low connectivity in the early stage of mining, thus achieving better performance for the high connectivity constraint. Our methods successfully mined interesting patterns from multiple biological networks. Through our study, we demonstrated the applicability of frequent graph mining in biological research.

9. REFERENCES

[1] C. Borgelt and M. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proc. 2002 Int. Conf. on Data Mining (ICDM'02)*, pages 211–218, 2002.

[2] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pages 443–452, 2001.

[3] A Butte, P. Tamayo, D. Slonim, T. Golub, and I. Kohane. Discovering functional relationships between rna expression and chemotherapeutic susceptibility. In *Proc. of the National Academy of Science*, volume 97, pages 12182–12186, 2000.

[4] C. Chekuri, A. Goldberg, D. Karger, M. Levine, and C. Stein. Experimental study of minimum cut algorithms. In *Proc. of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 324–333, 1997.

[5] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. In *Proc. of the National Academy of Science*, volume 95, pages 14863–14868, 1998.

[6] G. Flake, S. Lawrence, and C. Giles. Efficient identification of web communities. In *Proc. 2000 ACM Int. Conf. Knowledge Discovery and Data Mining (KDD'00)*, pages 150–160, 2000.

[7] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the subdue system. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 169 – 180, 1994.

[8] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining spatial motifs from protein structure graphs. In *Proc. of the 8th Annual Int. Conf. on Research in Computational Molecular Biology (RECOMB'04)*, pages 308–315.

[9] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. 2000 European Symp. Principle of Data Mining and Knowledge Discovery (PKDD'00)*, pages 13–23, 1998.

[10] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. 2001 Int. Conf. Data Mining (ICDM'01)*, pages 313–320, 2001.

[11] T. Mielikainen. Intersecting data to closed sets with constraints. In *Proc. of the First ICDM Workshop on Frequent Itemset Mining Implementation (FIMI'03)*, 2003.

[12] F. Pan, G. Cong, A. Tung, J. Yang, and M. Zaki. Carpenter: Finding closed patterns in long biological datasets. In *Proc. 2003 ACM Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, 2003.

[13] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[14] V. Spirin and L. Mirny. Protein complexes and functional modules in molecular networks. In *Proc. of the National Academy of Science*, volume 100, pages 12123–12128, 2003.

[15] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44:585–591, 1997.

[16] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, and T. Golub. Interpreting patterns of gene expression with self-organizing maps: methods and application to hematopoietic differentiation. In *Proc. of the National Academy of Science*, volume 96, pages 2907–2912, 1999.

[17] N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data. In *Proc. 2002 Int. Conf. on Data Mining (ICDM'02)*, pages 458–465, 2002.

[18] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proc. 2003 ACM Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, pages 236–245, 2003.

[19] T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explorations*, 5:59–68, 2003.

[20] D. West. *Introduction to Graph Theory*. Prentice Hall, Cambridge, MA, 2000.

[21] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15:1101–1113, 1993.

[22] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proc. 2002 Int. Conf. on Data Mining (ICDM'02)*, pages 721–724, 2002.

[23] X. Yan and J. Han. Closegraph: Mining closed frequent graph patterns. In *Proc. 2003 ACM Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, pages 286–295, 2003.

[24] X. Yan, P. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *Proc. 2004 ACM Int. Conf. Management of Data (SIGMOD'04)*, pages 335–346, 2004.

[25] M. Zaki and K. Gouda. Fast vertical mining using diffsets. In *Proc. 2003 ACM Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, pages 326–335, 2003.