A Direct Mining Approach To Efficient Constrained Graph Pattern Discovery

Feida Zhu
School of Information Systems
Singapore Management University
80 Stamford Road, Singapore
fdzhu@smu.edu.sg

Zequn Zhang
School of Computer Science and Technology
University of Science and Technology of China
zequn@acm.org

Qiang Qu
Department of Computer Science
Aarhus University
8200 Århus N, Denmark
qu@cs.au.dk

ABSTRACT

Despite the wealth of research on frequent graph pattern mining, how to efficiently mine the complete set of those with constraints still poses a huge challenge to the existing algorithms mainly due to the inherent bottleneck in the mining paradigm. In essence, mining requests with explicitly-specified constraints cannot be handled in a way that is direct and precise. In this paper, we propose a direct mining framework to solve the problem and illustrate our ideas in the context of a particular type of constrained frequent patterns — the "skinny" patterns, which are graph patterns with a long backbone from which short twigs branch out. These patterns, which we formally define as l-long δ-skinny patterns, are able to reveal insightful spatial and temporal trajectory patterns in mobile data mining, information diffusion, adoption propagation, and many others.

Based on the key concept of a *canonical diameter*, we develop SkinnyMine, an efficient algorithm to mine all the l-long δ-skinny patterns guaranteeing both the completeness of our mining result as well as the unique generation of each target pattern. We also present a general direct mining framework together with two properties of *reducibility* and *continuity* for qualified constraints. Our experiments on both synthetic and real data demonstrate the effectiveness and scalability of our approach.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; I.2.8 [**Problem Solving, Control Methods, and Search**]: Graph and tree search strategies

Keywords

Direct mining, frequent graph pattern mining, constrained pattern mining, skinny pattern

1. INTRODUCTION

The problem of mining frequent graph patterns is fundamental with many important applications including graph indexing [24, 25], graph clustering and classification [3], and graph query processing [2]. A wealth of mining algorithms have been proposed along the years [1, 9, 11, 12, 17, 20, 23], among which many of them handle various constraints on top of the frequency requirement [26]. However, the constrained graph pattern mining problem so far still poses a huge challenge to the existing algorithms mainly due to the inherent bottleneck in the mining paradigm. In essence, mining requests with explicitly-specified constraints cannot be handled in a way that is direct and precise. In this paper, we propose a direct mining framework to solve the problem and illustrate our ideas with a particular type of constrained frequent patterns — the "skinny" patterns. Recently, the importance of these skinny patterns has been increasingly recognized in heterogenous information network analysis [19] and social network studies.

A "skinny" pattern, formally called a l-long δ-skinny pattern, is essentially a graph pattern with a long backbone of length l from which short twigs no longer than δ branch out. It has found important applications for the descriptive power of its long backbone to represent spatial and temporal trajectories in heterogeneous information networks, and of the short twigs the various kinds of associated information, as illustrated in the following application examples.

Mobile data mining. In mobile applications with location based services and user-generated trajectory data and other content, e.g., foursquare and Instagram, skinny patterns can capture popular traveling routes and patterns (the backbone) together with the associated attributes/entities of interest (the twigs), e.g., nearby businesses, topics of user-generated-content, activities and so on. These patterns would offer useful knowledge for tasks such as recommendation, scheduling, clustering, etc.

Information diffusion analysis in social networks. In social network services, especially microblogging services like Twitter, skinny patterns mined from the re-tweeting network of tweets reveal insightful information diffusion patterns and different roles of users in the dissemination process. These patterns are key to network manipulation to either facilitate or hinder information diffusion [21].

Unfortunately, existing algorithms are not able to mine these skinny patterns efficiently. Essentially, algorithms exhausting all frequent patterns, gSpan [23] as a representative, are not able to find patterns larger than a threshold due to the exponential number of pattern candidates to enumerate and examine. Probabilistic algorithms returning a sample of the pattern output space, ORIGAMI [7] as a representative, would miss all but a few patterns of interest. The most relevant work is SpiderMine [26] designed for mining large patterns. However, due to the diameter bound constraint and the pattern growth based on "spiders", SpiderMine tends to find

SIGMOD'13, June 22–27, 2013, New York, New York, USA.

patterns that are large but "fat". As a result, as shown by our experiments, skinny patterns with long diameters but sparse twigs will often be missed.

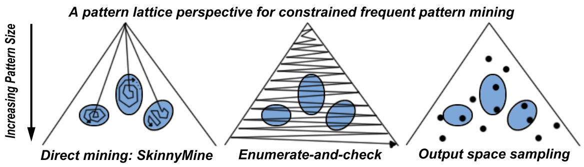
A pattern lattice perspective for constrained frequent pattern mining
Increasing Pattern Size
Direct mining: SkinnyMine
Enumerate-and-check
Output space sampling

Figure 1: A conceptual comparison among different mining paradigms

More importantly, similar gap for the mining tools challenges a large number of application settings where users are seeking the complete (or almost complete) set of patterns satisfying certain explicitly-specified constraints. What is ideal is a mining framework such that all target patterns can be reached fast (making it direct) with minimum visits to irrelevant ones (making it precise). As illustrated in Figure 1 from a pattern lattice perspective [28], the target mining result is "carved" by the constraint into pattern clusters indicated by the shaded areas. Probabilistic methods cannot accomplish the mining task because most of them are only capable of returning a sparse sample of the complete set of skinny patterns. Traditional "enumerate-and-check" approaches try to return all qualified results at the cost of exhaustively traversing the pattern space, failing usually halfway due to intractability. The direct mining approach, which we illustrate in this paper with SkinnyMine on the skinny pattern mining problem, provides a much more efficient way to reach straight some patterns in each of the target clusters and only examine locally those relevant candidates. Direct mining has been studied not on constraints on patterns but on certain utility of patterns such as discriminativeness measured on the mining results [4], which is a different problem from ours.

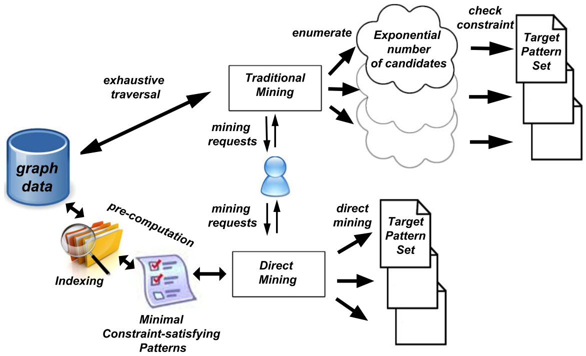

Figure 2: An architectural view of the direct mining framework

An architectural overview comparing our direct mining approach and traditional mining approach is depicted in Figure 2. Suppose user-specified mining requests are "find all skinny frequent patterns with diameter of length l" with varied l. Upon request with each l, traditional mining would make a almost exhaustive traversal of the pattern space, and every time enumerate an exponential number of frequent pattern candidates to check constraints before it finally generates the target pattern set. In contrast, direct mining works in a completely different way. It pre-computes all the minimal constraint-satisfying patterns, which we will define later, and build indexing structures to efficiently access the corresponding embeddings. Upon request with each l, it would directly access the proper minimal patterns associated with that l, and only examine locally all the relevant pattern candidates to generate the same target pattern set but much more efficiently.

Our Contributions. We propose a novel direct mining framework for efficient constrained graph pattern mining and demonstrate the idea and algorithms in the context of the skinny pattern mining problem.

First, we develop SkinnyMine, an efficient algorithm to mine all the l-long δ-skinny patterns. The algorithm is based on our proposed concept of *canonical diameters* that is important for both guaranteeing the unique generation of each target pattern as well as achieving a partition of all skinny patterns into disjoint result sets, thus enabling direct and precise mining. For example, suppose users are interested in finding all δ-skinny patterns with diameter length between l_1 and l_2 ($l_1 < l_2$), we are indeed able to mine exactly all the target patterns without even visiting those whose diameters are shorter than l_1 or longer than l_2.

Second, we formally prove that, by maintaining the canonical diameter through pattern extension, we can guarantee the completeness of our mining result. In order to efficiently maintain the canonical diameter, instead of invoking the expensive shortest path computation upon every edge extension, we decompose the task into maintaining three sufficient and necessary constraints, and propose and prove the correctness of a solution that only requires us to locally update two simple distance indices, significantly improving the mining efficiency.

Third, interestingly, to mine the canonical diameters, which are essentially frequent paths of length l, is not an easy task. We propose a novel and efficient mining algorithm based on the idea of progressively concatenating and merging existing paths of length in powers of 2, which gives us quick access to some patterns in the result pattern clusters.

An important contribution of this study is that we generalize the mining approach to propose a direct mining framework for a class of constrained frequent graph pattern mining problems. We present the direct mining framework and propose two properties of *reducibility* and *continuity* that a constraint should possess in order to apply the framework.

Road-map. The rest of the paper is organized as follows. We formulate our problem in Section 2. Section 3 presents the design ideas and overview of our approach. Algorithm details and implementation issues are discussed in Section 4. The direct mining framework is proposed in Section 5. Section 6 gives the experimental results. We discuss related works in Section 7 and conclude our paper in Section 8.

2. PROBLEM FORMULATION

As a convention, the *vertex set* of a graph G is denoted by $V(G)$ and the *edge set* by $E(G)$. Without specification, the size of a graph P is defined by the number of edges of P, written as $|P|$. In our setting, a graph $G = (V(G), E(G))$ is associated with a label function $l_G : V(G) \mapsto \Sigma$, where Σ is a label set. Graph isomorphism in our problem setting requires matching of the labels for each mapped pair of vertices. Our method can also be applied to graphs with edge labels.

DEFINITION 1. **(Labeled Graph Isomorphism)** *Two labeled graphs G and G' are isomorphic if there exists a bijection $f : V(G) \mapsto V(G')$, such that $\forall u \in V(G)$, $l_G(u) = l_{G'}(f(u))$ and $(u, v) \in E(G)$ if and only if $(f(u), f(v)) \in E(G')$.*

We use $G \cong_L G'$ to denote that two labeled graphs G and G'

are isomorphic. Given two graphs P and G, a subgraph G' of G is called an *embedding* of P in G if $P \cong_L G'$. For a single graph G and a pattern P, we use e_P to denote a particular embedding of a pattern P, and the set of all embeddings of P is denoted as $E[P]$.

Given a graph G, a path $\mathbb{L}$ of G is represented as a sequence of vertices $\mathbb{L} = [v_{i_1}, v_{i_2}, \dots, v_{i_k}]$ where $i_1, i_2, \dots, i_k$ are their physical vertex IDs and $(v_{i_m}, v_{i_{m+1}}) \in E(G), 1 \leq m < k$. By default, all paths in this paper are simple paths, i.e., all vertices are distinct. Without loss of generality, we assume there is a lexicographic order among all labels in Σ, and for any two labels $l_1, l_2 \in \Sigma$, denote as $l_1 \prec l_2$ if l_1 is ordered ahead of l_2. We first define a lexicographical order among paths.

DEFINITION 2. **[Lexicographical Path Order]** *For two labeled paths* $\mathbb{L}_1{=}[v^1_{i_1}, v^1_{i_2}, \dots, v^1_{i_{k_1}}]$ *and* $\mathbb{L}_2 = [v^2_{j_1}, v^2_{j_2}, \dots, v^2_{j_{k_2}}]$ *of a graph G with label function l, we say* $\mathbb{L}_1$ *is lexicographically smaller than* $\mathbb{L}_2$*, denoted as* $\mathbb{L}_1 \prec_L \mathbb{L}_2$*, if (I)* $k_1 < k_2$*; or (II)* $k_1 = k_2$ *and there exists an index* i^*, $1 \leq i^* < k_1$*, such that* $l(v^1_{i_m}) = l(v^2_{j_m})$ *for* $1 \leq m < i^*$*, and* $l(v^1_{i_{i^*}}) \prec l(v^2_{j_{i^*}})$*. We say* $\mathbb{L}_1$ *is lexicographically equal to* $\mathbb{L}_2$*, denoted as* $\mathbb{L}_1 =_L \mathbb{L}_2$*, if* $k_1 = k_2$ *and* $l(v^1_{i_m}) = l(v^2_{j_m})$ *for* $1 \leq m \leq k_1$.

With this we can define a total order among all paths of a graph. Essentially, for two given paths, we first compare their lexicographical order; and if they are lexicographically equal, we further compare their physical vertex ID sequences numerically. Formally, we have the following definition.

DEFINITION 3. **[Total Path Order]** *For any two labeled simple paths* $\mathbb{L}_1{=}[v^1_{i_1}, v^1_{i_2}, \dots, v^1_{i_{k_1}}]$ *and* $\mathbb{L}_2 = [v^2_{j_1}, v^2_{j_2}, \dots, v^2_{j_{k_2}}]$ *of a graph G with label function l, we say* $\mathbb{L}_1$ *is smaller than* $\mathbb{L}_2$*, denoted as* $\mathbb{L}_1 \prec \mathbb{L}_2$*, if (I)* $\mathbb{L}_1 \prec_L \mathbb{L}_2$*; or (II)* $\mathbb{L}_1 =_L \mathbb{L}_2$ *and there exists an index* i^*, $1 \leq i^* < k_1$*, such that for their physical ID sequences, we have* $i_m = j_m$ *for* $1 \leq m < i^*$*, and* $i_{i^*} < j_{i^*}$.

The *diameter* of a connected graph G is the maximum over the shortest distances between all pairs of vertices in $V(G)$, and is denoted as $\mathbb{D}(G)$. Given a connected graph G, let D_G be the set of all simple paths of length $\mathbb{D}(G)$ realizing the diameter. We define a *canonical diameter* of G as the smallest simple path in D_G. Formally, we have Definition 4.

DEFINITION 4. **[Canonical Diameter]** *Given a graph G of diameter* $\mathbb{D}(G)$*, let* D_G *be the set of all simple paths of length same with* $\mathbb{D}(G)$*. The* Canonical Diameter *of G, denoted as* $\mathbb{L}_G$*, is defined as* $\mathbb{L} \in D_G$ *such that for any* $\mathbb{L}' \in D_G$, $\mathbb{L} \prec \mathbb{L}'$.

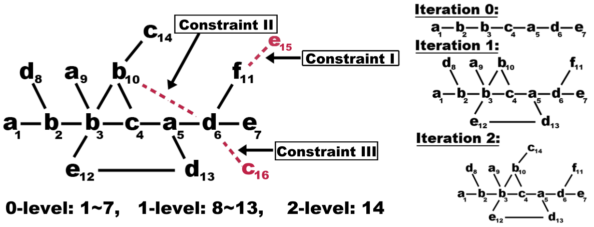
Constraint II
Constraint I
Constraint III
c_{14}
e_{15}
d_8 a_9 b_{10} f_{11}
a_1 b_2 b_3 c_4 a_5 d_6 e_7
e_{12} d_{13} c_{16}
0-level: 1~7, 1-level: 8~13, 2-level: 14
Iteration 0:
Iteration 1:
Iteration 2:

Figure 3: A 6-long 2-skinny graph (e.g., a_1 denotes the vertex with physical ID 1 and label 'a')

Let the canonical diameter of a current pattern P be $L = [v_H = v_1, v_2, v_3, \dots, v_{n+1} = v_T]$ with length $|L| = n$. Let v_H and v_T denote the *head* and *tail* of the diameter respectively. Figure 3 shows an example graph G with canonical diameter $\mathbb{L}_G = [v_1, v_2, v_3, v_4, v_5, v_6, v_7]$, the head $v_H = v_1$ and the tail $v_T = v_7$. Another path in the example is $\mathbb{L}' = [v_1, v_2, v_3, v_4, v_5, v_6, v_{11}]$ which is of the same length as $\mathbb{L}_G$ but is lexicographically larger by Definition 2. By Definition 3, each graph has a unique canonical diameter, which is the foundation for the unique pattern generation guaranteed in our framework.

For a graph G with canonical diameter $\mathbb{L}$ and a vertex $v \in V(G)$, let $Dist(v, \mathbb{L})$ denotes the shortest distance from v to $\mathbb{L}$, i.e., $Dist(v, \mathbb{L}) = \underset{Dist(v,u)}{\text{argmin}} \{Dist(v,u) | u \in V(\mathbb{L})\}$.

DEFINITION 5. **[Vertex Level]** *Given a graph G with canonical diameter* $\mathbb{L}$*, a vertex* $v \in V(G)$ *is called a d-*level *vertex if* $Dist(v, \mathbb{L}) = d$.

Next we define a δ-skinny graph, followed by a l-Long δ-skinny graph.

DEFINITION 6. **[δ-skinny Graph]** *A graph G is called* δ-skinny *if for all* $v \in V(G)$ *such that v is a d-level vertex, we have* $d \leq \delta$.

DEFINITION 7. **[l-Long δ-Skinny Graph]** *A graph G is called* l-long δ-skinny *if its canonical diameter* $\mathbb{L}$ *is of length l,i.e.,* $|\mathbb{L}| = l$*, and G is* δ-skinny.

Figure 3 presents an example of a 6-long 2-skinny graph G (in solid black edges) with vertices of each level shown at the bottom of the figure. Now we are ready to give our problem definition as follows.

DEFINITION 8. **[l-Long δ-Skinny Pattern Mining]** *Given a graph G, a frequency threshold* σ*, a length l and a skinniness bound* δ*, the problem of* l-Long δ-Skinny Pattern Mining $((l, \delta)$-SPM) *is to find all l-long* δ*-skinny subgraphs P of G such that* $|E[P]| \geq \sigma$.

Note that although our problem definition given above is under the single-graph setting, the corresponding version for graph transaction setting can be easily derived.

3. OUR APPROACH

3.1 Overview

As illustrated in Figure 1, to solve the (l, δ)-SPM problem with a direct mining approach, we identify two main challenges: (I) how to quickly reach certain patterns in the result set; and (II) Once inside in the result set, how to efficiently find the rest of the patterns. We have the following observation:

OBSERVATION 1. *Any frequent path L of length l is by itself a minimal pattern in the result set. On the other hand, any pattern P in the result set must contain a canonical diameter which is a frequent path of length l.*

This observation leads to the following two-stage algorithm:

Stage I: Mining Canonical Diameters.
Given a graph G, a frequency threshold σ and a user mining request with diameter constraint $l = l^*$, we mine all frequent simple paths of length l^*, and denote the result set as S_0. This gives us direct access to some patterns in the result set. Each such path is eventually the canonical diameter of all patterns grown from it.

Note that in a typical setting of our direct mining framework, we handle not just one mining request of a single l^*, but a sequence of mining requests with different constraint values of l^*. In this sense, canonical diameters are pre-computed as in Stage I and indexed by l^* with the corresponding embeddings.

Stage II: Growing Canonical Diameters to Skinny Patterns.
For each path $\mathbb{L} \in S_0$ such that $|\mathbb{L}| = l^*$, grow $\mathbb{L}$ in at most δ

iterations. In each iteration $i, 1 \leq i \leq \delta$, a current pattern P is extended by adding only two kinds of edges — (1) Edges connecting one $(i-1)$-level vertex and one i-level vertex; and (2) Edges connecting two i-level vertices.

Figure 3 shows the corresponding patterns we would obtain by the end of each iteration. Note that Iteration 0 is simply Stage I, i.e. vertices on the canonical diameter are 0-level.

We maintain the following important loop invariant for each iteration as Loop Invariant 1.

LOOP INVARIANT 1. *When growing a current pattern P with canonical diameter $\mathbb{L}$, $\mathbb{L}$ must remain the canonical diameter after each edge extension.*

By maintaining this loop invariant, we can guarantee the unique generation of each skinny pattern. To see this, first observe that the canonical diameters partition all l-long δ-skinny patterns into disjoint sets, all patterns in each set sharing the same canonical diameter. Within the same set, patterns are ordered lexicographically in a similar fashion as in gSpan [23], the details of which are omitted here due to space limit. Essentially, each pattern is encoded first by the number of levels and within the same level, "forward" edge and "backward" edges are defined similarly as in [23]. Each edge is encoded lexicographically by the labels of the two end vertices.

3.2 Mining Canonical Diameters

The challenge of mining canonical diameters is how to efficiently mine all the frequent paths of length l^*. Despite the much research work in frequent graph mining, few, if any, has focused on mining frequent simple paths to the best of our knowledge. Therefore we propose the following two-step algorithm.

Step I. First, we mine all the frequent paths whose lengths are powers of 2 and less than l^*, i.e., 2^0, 2^1, 2^2, ...,2^k, where $k = \underset{j}{\operatorname{argmax}}\{j|2^j \leq l^*\}$. To get paths of length 2^i, we concatenate two current frequent paths of length 2^{i-1} with the initial set being the set of all frequent edges of length 2^0. Compared against incremental edge extension, this approach provides the most efficient way to find all the paths with length of powers of 2, and we denote the set as S_p.

Step II. Next, we use S_p to find all frequent paths of length l where l is not a power of 2. We propose an efficient method that is based on the observation that any path of length l can be obtained by merging two paths of length 2^k where $k = \underset{j}{\operatorname{argmax}}\{j|2^j \leq l\}$. Note that in this case, in stead of concatenation, we merge two paths that are partially overlapping. The efficiency of this method is a result of the following two observations.

Firstly, any path L of length l can be *uniquely* obtained by merging two paths of length 2^k where $k = \underset{j}{\operatorname{argmax}}\{j|2^j \leq l\}$. This is because we have defined v_H and v_T as the head and tail of L, which are unique unless L is self-isomorphic. It follows that, if L is obtained by merging two paths, they must be the prefix of L containing v_H and the suffix of L containing v_T, both of which are of length 2^k.

Secondly, the fact that there are fewer frequent paths of longer lengths leads to the consequence that for longer paths, it is more efficient to obtain them by merging two longer sub-paths than concatenating a series of sub-paths of lengths of distinct powers of 2. For example, if L is of length 15, instead of concatenating 4 paths of lengths of distinct powers of 2, i.e. $15 = 8 + 4 + 2 + 1$, we only need to merge 2 paths each of length 8.

3.3 Maintaining Canonical Diameters

Given a current pattern P with canonical diameter $\mathbb{L}$, we have shown that as long as we maintain Loop Invariant 1, we would be able to guarantee the unique pattern generation. In Section 3.5, we will further prove the completeness of our mining result as a consequence of maintaining Loop Invariant 1. Now we first discuss how to efficiently maintain Loop Invariant 1. A naive way is to invoke all-pair shortest path computation after each edge extension. If Loop Invariant 1 is violated, we drop the pattern. This straightforward approach, however, is highly inefficient as all-pair shortest path computation is high costly for large graphs. Therefore we need to investigate smarter ways to maintain this constraint more efficiently.

Recall that v_H and v_T denote the *head* and *tail* of the diameter respectively. The key observation is that, when P is extended to P', to maintain that $\mathbb{L}$ is still the canonical diameter of P' is equivalent to maintaining the following three constraints:

1. **Constraint I: Diameter is not increased.**
   This means $\mathbb{D}(P') \leq \mathbb{D}(P)$, which guarantees that the edge extension does not create a longer diameter.

2. **Constraint II: $\mathbb{L}$ still realizes the shortest distance between v_H and v_T.**
   This means in P', $Dist(v_H, v_T) = |\mathbb{L}|$ and therefore $\mathbb{D}(P') \geq \mathbb{D}(P)$, which guarantees that the edge extension does not shorten the distance between v_H and v_T.

3. **Constraint III: $\mathbb{L} \prec \mathbb{L}'$ for any newly generated diameter $\mathbb{L}'$ of the same length.**

As indicated in Figure 3, we show (in red dotted lines) three examples of edge extension each violating one of the three above-mentioned constraints. For Constraint I, the extension would create a longer diameter of $[v_1, v_2, v_3, v_4, v_5, v_6, v_{11}, v_{15}]$ of length 7. For Constraint II, the extension would shorten $Dist(v_H, v_T)$ to 5 via the new edge. For Constraint III, the extension would create a new diameter $[v_1, v_2, v_3, v_4, v_5, v_6, v_{16}]$ with the same length as the current canonical diameter but smaller in lexicographical order.

It is worth noting that these three constraints are independent of one other, the satisfaction of any one of them does not guarantee the satisfaction of the other two. However, when maintained simultaneously, they are sufficient and necessary to guarantee Loop Invariant 1, i.e., the canonical diameter is preserved from P to P', as proved by the following lemma.

LEMMA 1. *Given a pattern P with its canonical diameter $\mathbb{L} = [v_H = v_1, v_2, v_3, \ldots, v_{n+1} = v_T]$ with length $|\mathbb{L}| = n$, for any pattern P' such that P' is obtained from P by one edge extension, $P \subset P'$ and $|E(P')| = |E(P)| + 1$, Constraint I, II and III are the sufficient and necessary conditions to guarantee that $\mathbb{L}$ is still the canonical diameter of P'.*

PROOF. We first show these three constraints are the sufficient conditions. The satisfaction of Constraint I means the diameter of P' is either the same as P or shorter. Constraint II, if satisfied, would then guarantee that the diameter of P' is not shorter than that of P since $\mathbb{D}(P') \geq Dist(v_H, v_T) = |\mathbb{L}| = n = \mathbb{D}(P)$. Therefore, Constraint I and II combined lead to the fact that $\mathbb{L}$ must be one of the diameter(s) of P'. Constraint III then further guarantees that, even the edge extension should create some new diameter $\mathbb{L}'$, we will have $\mathbb{L} \leq \mathbb{L}'$, making $\mathbb{L}$ the canonical diameter by definition. This completes the sufficient condition proof.

The necessary condition part is straightforward since if $\mathbb{L}$ is the canonical diameter of P', the diameter of P' is then the length of $\mathbb{L}$, satisfying Constraint I. There cannot be other shorter path between the head, v_H, and the tail, v_T, of $\mathbb{L}$ without violating the definition of a diameter. Constraint III is also trivial by Definition 4, which completes the necessary condition proof. □

3.4 Efficient Constraint Maintanence

Next we discuss how to efficiently maintain the three constraints. Recall that an i-level vertex is at distance i from the canonical diameter. So 0-level vertices would just be those of the canonical diameter itself. For each vertex u, we maintain two distances for u, D^u_H and D^u_T, which denote the shortest distances from u to v_H and v_T as the head and tail of the canonical diameter, respectively. In the following, we prove that we only need to locally update D^u_H and D^u_T to efficiently maintain all the three constraints.

3.4.1 Constraint I.

To maintain Constraint I, it turns out that we simply need to make sure that, when a new vertex u is added on the current pattern,

$$D^u_H \leq \mathbb{D}(P) \ and \ D^u_T \leq \mathbb{D}(P)$$

We need the following lemma before proving the correctness of the approach.

LEMMA 2. *Given a pattern P and any pattern P' such that P' is obtained from P by adding one edge (u, v) such that $v \in V(P)$ and $u \in V(P') \setminus V(P)$, we have $Dist(u, v) \leq \max(D^u_H, D^u_T)$ for any $v \in V(P)$.*

PROOF. We prove by induction on i, the iteration number of pattern growth. Recall that in the i-th iteration, only i-level vertices are added. For $i = 0$, the pattern P is simply the canonical diameter $\mathbb{L}$ itself. It is trivial to see that $Dist(u, v) \leq \max(D^u_H, D^u_T)$ for any $u, v \in V(P)$. Assuming the proposition holds for $i = n - 1$, we consider the case when $i = n$. In this case, the new edge connects u and v that are $(n - 1)$-level vertices. Since by induction $Dist(v, v') \leq \max(D^v_H, D^v_T)$ for any $v' \in V(P)$, and $Dist(u, v') = Dist(u, v) + Dist(v, v') = 1 + Dist(v, v')$ for any $v' \in V(P)$, we therefore have $Dist(u, v') = 1 + Dist(v, v') \leq 1 + \max(D^v_H, D^v_T) = \max(D^v_H + 1, D^v_T + 1) = \max(D^u_H, D^u_T)$. The proposition holds for $i = n$, which completes the proof. □

THEOREM 1. *Given a pattern P and any pattern P' such that P' is obtained from P by adding one edge (u, v), to guarantee that $\mathbb{D}(P') \leq \mathbb{D}(P)$, it is sufficient to guarantee that $D^u_H \leq \mathbb{D}(P)$ and $D^u_T \leq \mathbb{D}(P)$ (assuming u is the new vertex if any).*

PROOF. There are only two cases for the new edge (u, v): (1) $u \in V(P)$ and $v \in V(P)$, i.e., the new edge connects two existing vertices; and (2) $v \in V(P)$ and $u \in V(P') \setminus V(P)$, i.e., the new edge connects an existing vertex v with a new vertex u. In case (1), since connecting two existing vertices would only decrease the diameter, and we have $Dist(v', v'') \leq \mathbb{D}(P)$ for all vertex pair $v', v'' \in V(P)$, we must have $Dist(v', v'') \leq \mathbb{D}(P)$ for all vertex pair $v', v'' \in V(P')$. In case (2), first, for any existing vertex pair v', v'', $Dist(v', v'') \leq \mathbb{D}(P)$ still holds; we consider vertex pair u, v in which $v \in V(P)$ and $u \in V(P') \setminus V(P)$. Due to Lemma 2, the proposition holds as well. □

3.4.2 Constraint II.

To maintain Constraint II is to guarantee that the shortest distance between v_H and v_T is not reduced by the new edge. It is possible to first generate the pattern and then check if the resulting pattern satisfies Constraint II by running a breadth-first search with v_H as the root. However, a much more efficient way is make sure that, when a new vertex u is added on the current pattern,

$$D^u_H + D^u_T \geq \mathbb{D}(P)$$

The correctness is shown by the following theorem.

THEOREM 2. *Given a pattern P and any pattern P' such that P' is obtained from P by adding one edge (u, v), to guarantee that $\mathbb{D}(P') \geq \mathbb{D}(P)$, it is sufficient to make sure that $D^u_H + D^u_T \geq \mathbb{D}(P)$.*

PROOF. We prove by contradiction. Suppose for the purpose of contradiction that, even though it is guaranteed that $D^u_H + D^u_T \geq \mathbb{D}(P)$ for a new vertex u added during the growth iterations, $\mathbb{L}$ still is no longer the shortest path between v_H and v_T after adding edge (u, v). Let the new shortest path between v_H and v_T be $\mathbb{L}'$, and we have $|\mathbb{L}'| < |\mathbb{L}|$. Notice that the new shortest path $\mathbb{L}'$ must pass through this new vertex u, otherwise $\mathbb{L}$ will not be the shortest path even before adding u. Now consider the two sub-paths of $\mathbb{L}'$ divided by u, denoted as $\mathbb{L}'_1$ and $\mathbb{L}'_2$. By the property of submodularity of shortest paths, $\mathbb{L}'_1$ must be the shortest path between v_H and u, and $\mathbb{L}'_2$ must be the shortest path between u and v_T. Accordingly we have $|\mathbb{L}'_1| = D^u_H$ and $|\mathbb{L}'_2| = D^u_T$. Thus, we have $D^u_H + D^u_T = |\mathbb{L}'_1| + |\mathbb{L}'_2| = |\mathbb{L}'| < |\mathbb{L}| = \mathbb{D}(P)$, resulting in a contradiction and completing the proof. □

3.4.3 Contraint III.

To maintain Constraint III, it turns out that when P' is obtained from P by adding one edge (u, v), we only need to check whether $\mathbb{L} \prec \mathbb{L}'$ in the following two cases: (I) if $v \in V(P), u \notin V(P)$ and $\max(D^v_H, D^v_T) = \mathbb{D}(P) - 1$; and (2) if $u, v \in V(P)$ and either $D^u_H + D^v_T = \mathbb{D}(P) - 1$ or $D^v_H + D^u_T = \mathbb{D}(P) - 1$. In both cases, the new diameter $\mathbb{L}'$ passes through the new edge (u, v).

THEOREM 3. *Given a pattern P and any pattern P' such that P' is obtained from P by adding one edge (u, v), to guarantee that $\mathbb{L} \prec \mathbb{L}'$ for any newly generated diameter $\mathbb{L}'$, it is sufficient to check two cases: (I) if $v \in V(P), u \notin V(P)$ and $\max(D^v_H, D^v_T) = \mathbb{D}(P) - 1$; and (2) if $u, v \in V(P)$ and either $D^u_H + D^v_T = \mathbb{D}(P) - 1$ or $D^v_H + D^u_T = \mathbb{D}(P) - 1$.*

PROOF. We prove case (I), and case (II) can be obtained similarly. First, note that if both $u, v \in V(P)$, no new diameter would be created and since $\mathbb{L}$ is already the canonical diameter of P, Constraint III is satisfied automatically. It then follows that the newly generated diameter $\mathbb{L}'$ must contain the newly added vertex u as one of the end vertex. By Theorem 2, the other end vertex must be either v_H or v_T. Since the new diameter must be of length n, we have either $D^u_H = \mathbb{D}(P)$ or $D^u_T = \mathbb{D}(P)$. Therefore, the only case this happens is when we have $\max(D^v_H, D^v_T) = \mathbb{D}(P) - 1$. □

3.5 Proof of Completeness

We now prove why maintaining Loop Invariant 1 could guarantee the completeness of our mining result. To do that, we first observe that for each target pattern P, there exists a canonical diameter $\mathbb{L}$. As the l-long δ-skinny constraint possesses the property of *weak pattern anti-monotonicity* [27], P can be obtained through a pattern growth sequence of sub-patterns of increasing sizes, which all have $\mathbb{L}$ as their canonical diameter.

LEMMA 3. *Given a graph pattern P, there must exist a canonical diameter $\mathbb{L}$.*

PROOF. Immediate from Definition 3 and 4. □

LEMMA 4. *The constraint C of l-long δ-skinny is weak pattern-antimonotone [27], i.e., for a graph P where $|V(P)| \geq k$ for some constant k, $f_C(P) = 1 \to f_C(P') = 1$ for some $P' \subset P$ such that $|E(P')| = |E(P)| - 1$. In particular, for a l-long δ-skinny graph pattern P with $|E(P)| = m$, there exists a sequence of subgraph patterns $[P_l, P_{l+1}, \ldots, P_m = P]$ such that $P_l \subset P_{l+1} \subset$*

$\ldots, \subset P_m = P$ and $|E(P_i)| = |E(P_{i+1})| - 1, l \leq i < m$ where P_i denotes a pattern with $|E(P)| = i$.

PROOF. We give a constructive proof. Given a l-long δ-skinny graph pattern P with $|E(P)| = m$ and $\mathbb{L}$ as a canonical diameter, partition the set of vertices in $V(P) \setminus V(\mathbb{L})$ into at most δ disjoint sets according to their distance to $\mathbb{L}$ and order these sets in increasing order of the distances, denoting the result as $\Pi = [\pi_1, \pi_2, \ldots, \pi_j], 1 \leq j \leq \delta$. We construct the subgraph pattern sequence as follows. Start with $P_l = \mathbb{L}$. What follows is that at most δ sub-sequence of graphs ordered according to Π. In each sub-sequence $i, 1 \leq i \leq j$, each graph is obtained by adding to the preceding one one edge connecting the vertices in π_i or one vertex in π_i and one vertex in π_{i-1}. By Theorem 1, Theorem 2 and Theorem 3, the canonical diameter $\mathbb{L}$ will be maintained through this pattern growth process by edge extension. This means that all the graph patterns in the sequence satisfy the l-long δ-skinny constraint, proving its weak pattern-antimonotonicity. □

By Lemma 3 and Lemma 4, and the fact that we generate all frequent paths of length l in Stage I, we conclude that we mine the complete set of l-long δ-skinny graph patterns, which is formally stated as follows.

THEOREM 4. *Given a graph G, a frequency threshold σ, a length l, and a skinniness bound δ, SkinnyMine finds all l-long δ-skinny subgraphs P of G such that $|E[P]| \geq \sigma$.*

4. ALGORITHM DETAILS

In this section we give more detailed algorithm description. The main SkinnyMine algorithm, as described in high level in Section 3, is shown in Algorithm 1. The two subroutines DiamMine() and LevelGrow() are detailed in Algorithm 2 and Algorithm 3, respectively.

Algorithm 1 SkinnyMine

```
Input: input graph $G$, length constraint $l^*$, skinny threshold $\delta$
        support threshold $\sigma$
Output: Set of all qualified patterns $S$
1: $S \leftarrow \emptyset$;
2: If $S_0 = \emptyset$
3:        $S_0 \leftarrow$ DiamMine$(G, l^*, \sigma)$;
4: For  each $L \in S_0$
5:        For  $i = 1$ To $\delta$
6:             $S_i \leftarrow \emptyset$;
7:             For  each pattern $P \in S_{i-1}$
8:                  $S_i \leftarrow S_i \bigcup$ LevelGrow$(G, P, \sigma)$;
9:             If  $S_i \neq \emptyset$
10:                 $S \leftarrow S \bigcup S_i$;
11:            Else
12:                 Break;
13: Return  $S$;
```

DiamMine() is the routine to find all frequent paths of length exactly l^*, which, more typically, would be invoked only once in the pre-computing stage of our direct mining framework. S_0 is assumed to be a global structure such that SkinnyMine can access upon each mining request. Line 2 to Line 7 is the part where we find all frequent paths whose lengths are powers of 2 by progressively concatenating current paths. Line 10 to Line 17 finds all paths of length l^* where l^* is not a power of 2 by using the paths found in the first part. Notice that in CheckConcat() we only check if we can concatenate two paths by indexing their two end-vertices. In CheckMergeHead() and CheckMergeTail(), we merge two paths of length k by overlapping them. CheckMergeHead() checks the case when L' forms the head of the resulting path while CheckMergeTail() checks the case when L' forms the tail. Note that DiamMine() can be adapted to return frequent paths of length at least l^* with minor changes.

Algorithm 2 DiamMine

```
Input: input graph $G$, length constraint $l^*$, support threshold $\sigma$
Output: Set of all qualified frequent paths $S$
1: $S \leftarrow \emptyset, i \leftarrow 0, S^0 \leftarrow$ all frequent edges;
2: Do until $2^i > l^*$
3:      $i \leftarrow i + 1$;
4:      For  each pair of paths $L', L'' \in S^{i-1}$
5:           $\hat{L} \leftarrow$ CheckConcat$(L', L'')$;
6:           If $\hat{L}$ is frequent
7:                $S^i \leftarrow S^i \bigcup \{\hat{L}\}$;
8: $T \leftarrow S^i$;
9: If  $l^* > 2^i$
10:     For each $L' \in T$
11:          For each $L'' \in T$
12:               $\hat{L} \leftarrow$ CheckMergeHead$(L', L'')$;
13:                    If $\hat{L}$ is frequent and $|\hat{L}| = l^*$
14:                         $S \leftarrow S \bigcup \{\hat{L}\}$;
15:               $\hat{L} \leftarrow$ CheckMergeTail$(L', L'')$;
16:                    If $\hat{L}$ is frequent and $|\hat{L}| = l^*$
17:                         $S \leftarrow S \bigcup \{\hat{L}\}$;
18: Else $S \leftarrow T$;
19: Return  $S$;
```

Algorithm 3 LevelGrow

```
Input: input graph $G$, current pattern $P$, support threshold $\sigma$
Output: the set of extended patterns $S$
1: $S \leftarrow \emptyset, T_{pre} \leftarrow \{P\}$;
2: $E \leftarrow$ all possible edges for extension in canonical order;
3: While  $T_{pre} \neq \emptyset$
4:      $T_{current} \leftarrow T_{pre}$;
5:      $T_{pre} \leftarrow \emptyset$;
6:      For  each pattern $P \in T_{current}$
7:           For  each edge $(u, v) \in E$ ordered after $P_{anchor}$
8:                $P' \leftarrow P \bigcup (u, v)$;
9:                Update $D_H$ and $D_T$ for $u$ and $v$;
10:                If CheckConstraint$(P')$
11:                     If $P'$ is frequent
12:                          If $P'$ is closed, Then $S \leftarrow S \bigcup P'$;
13:                          $T_{pre} \leftarrow T_{pre} \bigcup \{P'\}$;
14:                          $P_{anchor} \leftarrow (u, v)$;
15: Return  $S$;
```

Given a pattern P such that the maximum distance from a vertex to its canonical diameter is d, then LevelGrow() is the routine to grow P to a set of super-patterns by adding only edges either connecting to new $(d+1)$-level vertices from d-level or connecting two $(d+1)$-level vertices. Note that all edges qualified for extension in each iteration are ordered lexicographically and a pattern records in P_{anchor} the last edge added on to it so far, so that next time it will only be extended with edges that are ordered after P_{anchor}. In Line 10, we check all the three constraints as elaborated in Section 3. Note that in order to do that, we only need to check D_H and D_T locally, which are two indices stored with each vertex and can be updated incrementally when needed.

5. A DIRECT MINING FRAMEWORK

In this section we generalize our approach in the skinny pattern mining setting to present a direct mining framework for a class of constrained frequent graph pattern mining problems, and discuss the properties that a constraint should have in order to apply the framework.

5.1 Framework

The general problem description for constrained frequent graph pattern mining is the following:

DEFINITION 9. *Given a single graph or a graph transaction database D, a minimum frequency threshold σ, and a specified constraint C, find all frequent subgraph patterns such that each pattern satisfies C.*

Note that the constraint C represents a general constraint that is not limited to a single constraint and could be a conjunction of a set of atomic constraints. As illustrated in Figure 2, our direct mining framework for the general mining problem works in two stages:

1. ***Minimal Constraint-satisfying Pattern Generation.***
   This stage of mining the minimal constraint-satisfying patterns can often be pre-computed off-line. The corresponding embeddings of each minimal pattern can be then indexed for efficient access later.
2. ***Constraint-preserving Pattern Growth.***
   Upon user request for mining with a particular constraint, the corresponding minimal patterns can be fetched and the proper embeddings are retrieved from graph database. Next the constraint-preserving pattern growth stage will grow each minimal pattern to find all target patterns.

In order to apply the direct mining framework, the constraint of interest needs to have certain properties, namely *reducibility* and *continuity*, which we detail as follows.

5.2 Minimal Constraint-satisfying Pattern

The first stage of our mining framework is to find certain "anchor" pattern node to start the mining process. As we are adopting a pattern-growth paradigm, the best choice is to find the minimal pattern that satisfies the constraint. In our skinny pattern problem setting, if the constraint is being l-long δ-skinny, then the minimal constraint-satisfying patterns are the frequent paths of length l, which is why we mine them first. The importance of this stage is that, due to the reduced pattern size and simplified constraint, it is usually possible to develop more efficient algorithms customized for mining these minimal patterns than using general-purpose mining algorithms to get them by enumerate-and-check.

To enable this stage, the given constraint must admit the existence of such minimal patterns of non-trivial size, e.g. it cannot be a single vertex. Formally, we have the following property. Given a constraint C, we denote as $f_C(.)$ the boolean predicate defined on the pattern space such that for a pattern P, $f_C(P) = 1$ if and only if P satisfies constraint C.

PROPERTY 1. ***[Reducibility]*** *A graph constraint C is called* reducible *if there exists a positive integer k and a nonempty set S of graph patterns such that for each $P \in S$, $|E(P)| \geq k$ and $f_C(P) = 1$, and for any pattern $P' \subset P$, $f_C(P') = 0$.*

Note that not every graph constraint is reducible. Consider the constraint of $MaxDegree(G) \leq K$, i.e., the maximum node degree is smaller than K. It is easy to verify that there is no minimal constraint-satisfying patterns except for the trivial single vertices. This means that no efficient algorithm can be proposed to mine a targeted subset of the whole pattern space in order to reduce pattern enumeration. Indeed, in this case, all frequent patterns must be examined because the constraint does not define a subset of patterns of non-trivial size.

5.3 Constraint-preserving Pattern Growth

The second stage of our mining framework is to grow each minimal constraint-satisfying pattern generated at the first stage while still preserving the constraint during growth. Again as in our skinny problem setting, we preserve the constraint by making sure that the canonical diameter remains to be the canonical diameter after each edge extension. Will we miss any target pattern in this way? In order to mine the complete pattern set, the constraint must admit the following property, which essentially states that the target patterns within one pattern cluster defined by a minimal pattern should be adjacent to some subpatterns in the same cluster.

PROPERTY 2. ***[Continuity]*** *A graph constraint C is called* continuous *if for each pattern P such that $f_C(P) = 1$, one of the following is true: (1)P is a minimal constraint-satisfying pattern; (2)there exists at least one pattern P' such that $P' \subset P$, $|E(P')| = |E(P)| - 1$ and $f_C(P') = 1$.*

This property guarantees that we will not miss any target pattern by growing from the minimal constraint-satisfying patterns. Note that not every graph constraint is continuous. Consider the constraint that "$Degree(v) = Degree(v')$ for all vertex pair $v, v' \in V(P)$", i.e., each vertex of the graph has the same degree. It is easy to verify that this constraint is not continuous.

6. EXPERIMENTAL RESULTS

In this section, we present our performance study on SkinnyMine. We first present our evaluation methodology.

6.1 Evaluation Methodology

We use both synthetic data and real data for our evaluation. On synthetic data, we systematically evaluate SkinnyMine for both effectiveness and scalability (Section 6.2). On real data sets, we present interesting pattern examples in our mining result to demonstrate the application of skinny patterns (Section 6.3).

6.1.1 Effectiveness

We demonstrate in Section 6.2.1 that SkinnyMine is able to mine out skinny patterns that cannot be discovered by other algorithms. We examine both graph-transaction setting and single graph setting, and compare against the state-of-the-art algorithms. In each setting, we first compare SkinnyMine against existing algorithms under various data settings to show the difference in the patterns obtained. Due to the inherent bottleneck in existing algorithms, these comparisons have to be constrained to graphs with relatively small sizes to allow all competing algorithms to run to completion. When the limits of other algorithms are reached, we further demonstrate the performance of SkinnyMine by itself.

6.1.2 Scalability

To demonstrate the scalability of SkinnyMine, we show the following in Section 6.2.2: (I) we compare SkinnyMine with existing algorithms; (II) we present the runtime of SkinnyMine against varied constraints, i.e., pattern diameter l and pattern skinniness δ.

Experiments in this part also illustrate the impact of the two properties of *Reducibility* and *Continuity* of our direct mining frame-

GID	$\|V\|$	f	deg	$\|V_L\|$	L_d	L_s	n	$\|V_S\|$	S_d	S_s
1	500	80	2	40	18	2	5	3	2	2
2	500	80	4	40	18	2	5	3	2	2
3	1000	240	2	40	18	2	5	3	2	20
4	1000	240	4	40	18	2	5	3	2	20
5	600	150	4	40	18	2	20	3	2	2

Table 1: Data settings.

GID vs GID	difference in setting
2 vs 1	GID 2 doubles the average degree
3 vs 1	GID 3 increases the support of short patterns.
4 vs 3	GID 4 doubles the average degree
5 vs 2	GID 5 increases the number of short patterns.

Table 2: Setting difference.

work by showing the break-down of the runtime for the two corresponding mining stages.

All experiments were conducted on an Intel Core i7 2.7 GHz CPU with 8GB main memory and running Ubuntu 12.04. Our algorithm SkinnyMine was implemented in C++ using Standard Template Library, while the other algorithms (SUBDUE (version 5.2.2), SEuS (version 1.0), MoSS (version 5.3), SpiderMine, and ORIGAMI) are all obtained from their original authors to whom we are greatly thankful. The algorithms are carefully chosen as they represent the state of the art in this area, and readers are referred to [26] for detailed descriptions for each.

6.2 Synthetic Data

6.2.1 Effectiveness

Single-Graph Setting. As SpiderMine [26] is the closest work to ours which aims to find large patterns in a single-graph setting, we have intentionally set our data settings similar to those in [26] such that performance of the two algorithms can be more easily compared. SkinnyMine is experimented on single graphs generated with the well-known Erdős-Rényi random network model. Using the $G(n, p)$ variant, our synthetic single graphs are constructed by generating a background graph and injecting into it long (or short) skinny patterns. Five different data sets (labeled GID 1 to 5) were generated with varied parameter settings. The detail descriptions of the five data sets are given in Table 1. The differences among the data sets are described in Table 2. The description of the parameters is given as follows: $|V|$ is the number of vertices in the graph. f is the number of different vertex labels. deg is the average degree. m (or n) is the number of injected long (resp. short) patterns ($m = 5$ for all the five data sets and is omitted in the table). $|V_L|$(or $|V_S|$) is the number of vertices of each injected long (resp. short) patterns. L_d (or S_d) is the length of the diameter of each injected long (resp. short) patterns. L_s (or S_s) is the number of embeddings of each injected long (resp. short) patterns.

The scale of the synthetic data, e.g., the number of vertices, the average degree, has been purposely set small so that all the three algorithms (SUBDUE, SEuS and SpiderMine) are able to run to completion.

Figures 4 to 8 show the distribution of patterns mined by SUBDUE, SEuS, SpiderMine and SkinnyMine for different parameter settings in Table 1. Overlapping bars indicate the same pattern size. To find as many patterns as possible, the minimum support threshold has been set to a small value of 2 in all these cases and the skinniness bound δ is not specified. We have following observations.

1. **SkinnyMine.** Note that *in all 5 cases, SkinnyMine successfully finds all the injected long patterns of diameter at least 18.* In GID 2, after 5 patterns of size 40 have been explicitly embedded into the background graph, the interconnections between the patterns and the background graph actually give rise to 3 patterns of size 41.
2. **SpiderMine.** We set $K = 5$ and $D_{max} = 4$. SpiderMine is able to find most of the injected patterns when we set the number of initial spiders to be picked to a fairly large value of 200. However, even at the cost of run time as a result of picking a large number seed spiders, SpiderMine misses the longest patterns in GID 1, GID 2 and GID 3.

   To further demonstrate that SkinnyMine can find skinny patterns that SpiderMine will miss, we conduct the following experiment. We generate a graph with $2,000$ vertices, $deg = 3$ and $f = 100$, and inject 10 different patterns each with support 2. As shown in Table 3, the 10 patterns are designed to be of decreasing degree of "skinniness" with PID 1 being the most skinny among the first 5 graphs (PID 1 to PID 5) and PID 6 being the most skinny among the last 5 graphs (PID 6 to PID 10), and the first 5 graphs are more skinny than the last 5 graphs as a whole. The result is that SkinnyMine captures the most skinny ones (PID=1, 2, 3, 4, 5) while SpiderMine finds three least skinny ones and two medium skinny ones (PID=4, 5, 8, 9, 10).
3. **SUBDUE.** SUBDUE focuses on small patterns with relatively high frequency. In Figures 6 and 7 confirmed results in [26], when the support of each small patterns increases, the mining result of SUBDUE shifts significantly toward smaller patterns. The same outcome can be observed in Figure 8 when the number of small patterns increases.
4. **SEuS.** SEuS adopts a node collapsing heuristics, which is less powerful in handling a large number of patterns with low frequency. In our experiments, like results shown in [26], SEuS has mostly generated small patterns ($|V| <= 3$) across the five data sets.
5. **MoSS.** Since MoSS aims to mine the complete pattern set, its runtime complexity is high. Although it could generate all frequent patterns in theory, in our experiments, MoSS cannot run to completion for data sets with $GID = 2, 4, 5$ within 5 hours as shown in Figure 20.

Graph-Transaction Setting. Despite the fact that our SkinnyMine is developed for the single-graph setting, we show that it can also be adapted to apply to the traditional graph-transaction setting. We construct a graph database by generating 10 graphs by Erdős-Rényi random network model each with 800 vertices, $deg = 5$, and $f = 80$. We then inject 5 distinct skinny patterns each with 40 vertices, a diameter of 20, and a support of 5 embeddings. In Figure 9, we compare SkinnyMine against SpiderMine and ORIGAMI for the distribution of the patterns obtained. Observation is that SkinnyMine gets mostly the largest patterns while SpiderMine finds those of medium to large sizes, and ORIGAMI returns a scattered sample composed of a few medium-sized patterns and mostly small ones. Next, we inject another 120 small patterns each of 5 vertices and a support of 5 into the data set, and run the algorithms again. This time, as shown in Figure 10, while SkinnyMine and SpiderMine still find the larger patterns, ORIGAMI finds mostly the small ones and misses even the medium-sized ones. This further confirms that both SpiderMine and ORIGAMI are not suitable for mining skinny patterns.

PID	1	2	3	4	5	6	7	8	9	10
|V|	60	60	60	60	60	20	30	40	50	60
Diameter	50	45	40	35	30	8	8	8	8	8

Table 3: 10 Graphs of varied skinniness.

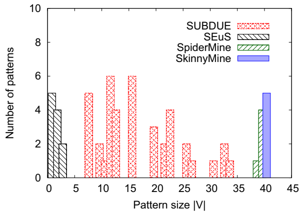

Figure 4: GID 1.

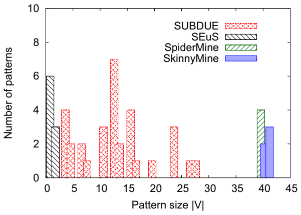

Figure 5: GID 2.

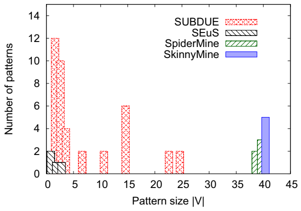

Figure 6: GID 3.

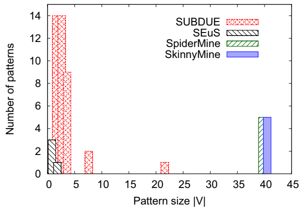

Figure 7: GID 4.

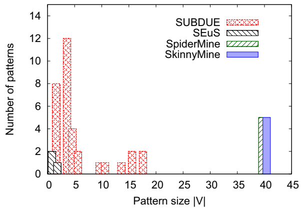

Figure 8: GID 5.

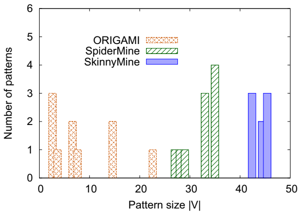

Figure 9: Fewer smaller patterns injected.

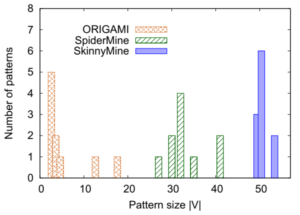

Figure 10: More smaller patterns injected.

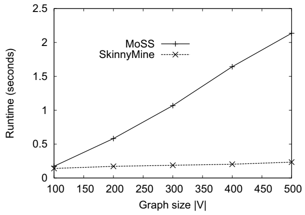

Figure 11: Runtime vs MoSS.

Run Time (seconds)					
GID	SkinnyMine	SpiderMine	SUBDUE	SEuS	MoSS
1	0.312	0.335	0.34	0.812	2.196
2	0.647	2.634	6.12	92.574	> 18000
3	0.449	0.732	2.77	1.297	2.683
4	0.933	10.763	15.64	874.251	> 18000
5	0.440	5.322	8.26	1.724	> 18000

Figure 20: Runtime comparison.

6.2.2 *Scalability*

Comparisons. Since MoSS aims to mine the complete pattern set, its runtime complexity is high. In Figure 20, we show the run time of the 5 algorithms on the 5 data sets. MoSS cannot run to completion for data sets with $GID = 2, 4, 5$ within 5 hours. In order to compare the runtime between MoSS and SkinnyMine, we decrease the average node degree to 2 ($deg = 2$, $f = 70$) in Figure 11 so that MoSS can finish execution.

In Figure 12 and Figure 13,we compare with SUBDUE and SpiderMine for runtime with $deg = 3$, label number $f = 100$, and support threshold set to 2. The findings show that the runtime of SkinnyMine grows much slower than that of SUBDUE and SpiderMine. For SpiderMine, we set $K = 10$ so that SpiderMine would return the 10 largest patterns, while SkinnyMine aims to find the skinny patterns with the longest diameter by growing all the skinny patterns from the longest canonical diameters.

In Figure 14, we show the scalability of SkinnyMine on much larger graphs up to 300,000 nodes with $deg = 3$ and $f = 80$. The mining task is to find all frequent l-long δ-skinny patterns with $l \geq 4$ and $\delta = 3$. The support threshold is set to 2. We show in Figure 14 the running time for Stage I DiamMine and Stage II LevelGrow, respectively. The corresponding number of patterns are shown in Figure 15.

Constraints. We show how the user-specified constraints could affect the scalability of SkinnyMine. We also demonstrate the power of the properties of *Reducibility* and *Continuity* of our direct mining framework.

We first examine the diameter constraint l. In Figure 16 and Figure 17, we show the runtime of the two stages of SkinnyMine respectively together with the number of patterns found in the corresponding stage. We set the support threshold as 2, pattern skinniness $\delta = 2$, $|V| = 10,000$, $deg = 3$, $f = 10$, and the longest diameter found is of length 19. Figure 16 shows the runtime of the DiamMine stage as l increases from 2 to 18. Recall that for each value l_c of l, our aim in this stage is to find all frequent paths of length exactly l_c, each of which corresponds to a canonical diameter to be further grown in the second stage. As shown in the figure, there are significantly more shorter frequent paths than longer ones, e.g., there are 549 distinct frequent paths of length 2, $30,871$ of length 4, $2,842$ of length 8, and only 3 of length 16.

The growth of the runtime of DiamMine can be best explained by the way we use frequent paths of length 2^k already found to obtain longer ones, as detailed in Section 3.2. The step growth from $l = 5$ to $l = 8$ is due to the following two reasons: (1) The huge number of paths of length 4 that are used to obtain these longer paths means in general we have a large number of candidates to check, and (2) As the target diameter length increases, the overlapping part of two length-4 paths merged to obtain the target diameter decreases. For example, to obtain the target length 5, the overlapping part of two length-4 paths is of length 3, whereas for target length 6, the overlapping part is then of length 2. The weaker constraint means we have more candidates to check as l increases. The almost flat growth curve from $l = 8$ onwards is due to the small number of length-8 paths that are used to obtain these longer ones, which is consistent with the general intuition that the truly long frequent paths are rare. Notice that the runtime includes the time to first obtain the length-8 ones. Hence the plateau shape.

The plateau-shaped growth curve demonstrates the power of the *Reducibility* property of our direct mining framework. The fact that we can efficiently mine all canonical diameters of various sizes (even for the longer ones) means that, by exploiting the reducibility property of the constraint, we have successfully avoided being trapped in the exponential number of irrelevant patterns and directly reach the minimal patterns in the mining result of interest.

Figure 17 shows the runtime of the LevelGrow stage as l increases from 2 to 18. Recall that for each value l_c of l, our aim in this stage is to grow each canonical diameter of length l_c to all l_c-long d-skinny patterns. The distribution of the number of patterns is also intuitive, e.g., there are $2,931$ distinct patterns for $l_c = 2$, $36,956$ for $l_c = 4$, $4,258$ for $l_c = 8$, and only 16 for $l_c = 16$.

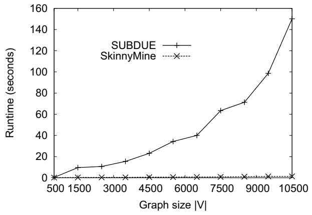

Figure 12: Runtime vs SUBDUE.

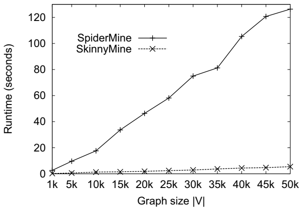

Figure 13: Runtime vs SpiderMine.

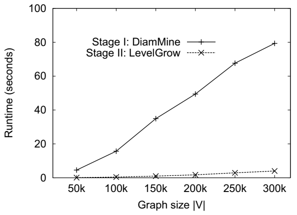

Figure 14: Scalability: Runtime.

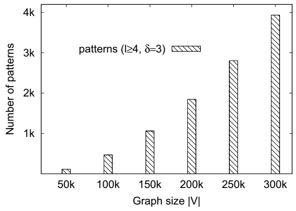

Figure 15: Scalability: # of patterns.

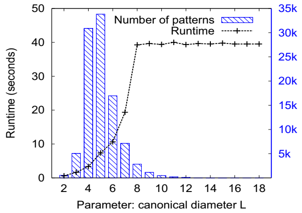

Figure 16: Runtime of DiamMine with # of patterns.

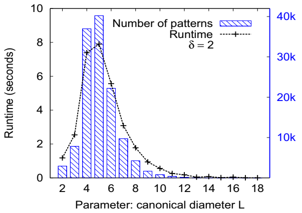

Figure 17: Runtime of LevelGrow with # of patterns.

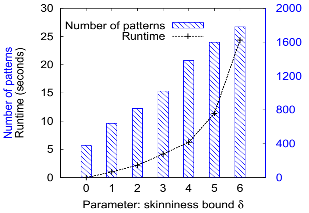

Figure 18: Runtime of LevelGrow with # of patterns.

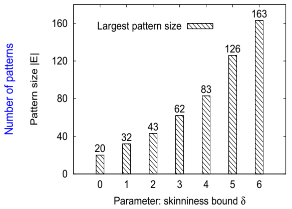

Figure 19: Maximal pattern sizes.

It can be observed that the runtime of LevelGrow is linear to the number of patterns in the final result, illustrating the effectiveness of Constraint I, II, and III for efficient canonical diameter maintenance as well as our techniques for efficient maintenance of these three constraints themselves.

The fact that the runtime of LevelGrow is linear to the number of patterns in the final result demonstrates the power of the *Continuity* property of our direct mining framework. Once the minimal constraint-satisfying patterns are found, all constraint-satisfying patterns can be found efficiently by examining locally those adjacent to the minimal patterns, without checking all the irrelevant candidates in the search space.

We next examine the skinniness constraint δ. Since the DiamMine stage would be the same for all varied δ (which costs 41 seconds), in Figure 18 we show the runtime of the LevelGrow stage as the skinniness threshold δ increases from 0 to 6, with the diameter constraint fixed at $l = 20$. The data graph is generated with $|V| = 200,000$, $deg = 3$, $f = 100$, and 250 distinct injected patterns each with $l = 20$, $\delta = 6$, $|V| = 50$, and 5 embeddings. Figure 19 shows the largest size of patterns mined for each δ. It can be observed that the growth of runtime is linear up to $\delta = 4$ and the jump from $\delta = 5$ onwards is due to the significant increase in the largest found pattern size as illustrated in Figure 19. In real applications, δ is usually a small number, e.g., 2 or 3.

6.3 Real Data

DBLP.

DBLP (http://www.informatik. uni-trier.de/∼ley/db/) provides bibliographic information on major computer science journals and proceedings. As of November 2012, DBLP data contains more than $1,177,381$ papers from $808,657$ distinct authors and $3,285$ conferences. We count 263 top Database conferences provided by Microsoft Academic Research [1], and then pick $86,443$ authors who published or coauthored at least one paper in these 263 Database conferences. We construct from this DBLP data a graph dataset of $9,363$ graphs in which each graph is a heterogeneous network and corresponds to one distinct author. In particular, the graph is composed of one continuous time-line where each node represents one year. Each such node is connected to at most four types of author nodes defined as follows.

First we give 4 types of labels to each author as: An author is assigned a label "Prolific (P)" if the author published at least 50 papers in the area of Database. The "Senior (S)" label is assigned to authors with 20 to 49 papers; "Junior (J)" with 10 to 19 papers and "Beginner (B)" with 5 to 9 papers. The authors with fewer than 5 papers are not considered. Note that when counting, we look at the number of publications up to the particular year of interest. Second, for two authors A and B, we say they *collaborate* in year i if they co-author at least one paper in that particular year. Third, for each author A, we classify A into three levels of collaboration strength (Level 1, 2, and 3 respectively) with an author category (P, S, J, and B) by how many authors of that category A has collaborated in that year. The levels are defined as follows: Level 1: if A collaborates with 1 to 2 authors of that particular category in that year, Level 2: 3 to 4 authors, and Level 3: 5 authors and above. Finally, A is connected to a node with label in the form of "X_k" where $X \in \{P, S, J, B\}$ and $k \in \{1, 2, 3\}$ according to the above requirements. For example, if A collaborates with 4 "Senior" authors in that year, A will be connected to a node labeled "S_2".

We run SkinnyMine on the resulting graph dataset with frequency threshold as 2 and length constraint at least 19, which means we are only interested in patterns across 20 years and above. Altogether SkinnyMine has found $84,273$ skinny frequent patterns whose diameter is at least 19 in 947 seconds. There are $1,315$ distinct authors who have at least a 20-year DBLP timeline. We give some pattern examples we have found in the dataset.

Pattern Example. We show two temporal collaborative patterns each across a time-span of 20 years of the researchers' publication records in DBLP. Note that the exact years are different in each embedding. Figure 21 shows a temporal collaborative pattern and the corresponding publication years shared by Jiawei Han (1993 to 2012) and Philip S. Yu (1991 to 2010). This pattern shows that these researchers have collaborated with an increasing number of more productive authors along their research career. Figure 22 shows a different temporal collaborative pattern and the corresponding publication years with the three embeddings being Di-

[1] http://academic.research.microsoft.com

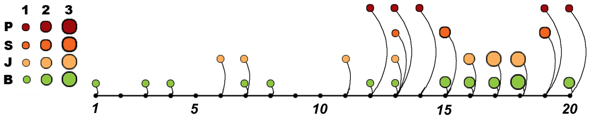

Figure 21: DBLP skinny pattern example 1: A temporal collaborative pattern across 20 years.

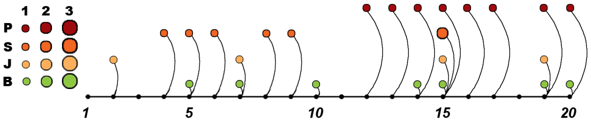

Figure 22: DBLP skinny pattern example 2: A temporal collaborative pattern across 20 years.

vyakant Agrawal (1991 to 2010), Amr El Abbadi (1991 to 2010) and Jeffrey F. Naughton (1988 to 2007). This pattern shows that, since their early research career, these researchers have already collaborated with some fairly productive authors.

Sina Weibo.
Sina Weibo is a popular, Twitter-like micro-blogging service platform originated from China [2]. It is reported to have a registered total user base of 358 million, of which roughly 36.5 million active users daily. Two important features that are not yet available on Twitter are (I) a user can comment on any other user's tweets, which results in more user interaction; and (II) the retweeting chain is visible to the public, which is critically important for studying the information diffusion process and user interaction pattern.

We use a Weibo dataset consisting of more than 1.8 million users and 230 million tweets, of which 111 million are original tweets. From this dataset, we create a graph dataset for all the popular tweets that have been retweeted more than 100 times. We define a *conversation* as the graph formed with the author of the original tweet as the root, and each retweet and comment would add an edge between the user performing the action and the target user. Note that a user could appear multiple times within the same conversation. All users are assigned one of three labels: (1) the root user, i.e., author of the original tweet, (2) the users who follow the root user, and (3) all other users. A conversation example is shown in Figure 23, in which the nodes in red represent the root user. We set the length constraint as at least 10 as we aim to find long diffusion paths, and frequency threshold as 2. Out of a resulting graph dataset of a total number of $2,236,424$ vertices and $1,978,311$ edges, SpiderMine mined $13,847$ frequent skinny patterns in 806 seconds.

Pattern Example. Figure 24 shows an example of a frequent interaction pattern mined from the conversations. The skinny diffusion chain is a 13-long 3-skinny pattern. The arrow denotes the direction that the information flows, i.e., the direction in which the tweet gets disseminated. The red nodes are the same root user who originates the tweet, and the green nodes are all her followers. This pattern reveals two interesting insights. First, the root user is the type of micro-bloggers who enjoy interacting with her followers and would indeed engage in dialogues with them as her tweets get passed on. Second, each time the root users engage in the conversation, the tweet get further promoted to reach a larger audience.

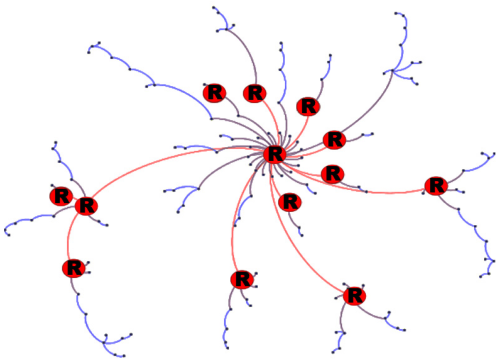

Figure 23: A conversation example in Sina Weibo

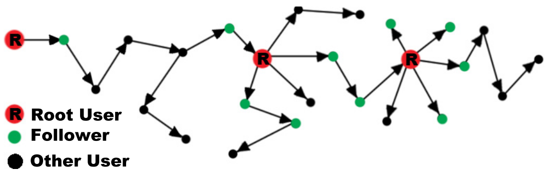

Figure 24: Weibo: An interaction pattern in conversations

7. RELATED WORK

Many efficient algorithms have been developed to find frequent patterns in graph transactions, e.g., AGM by Inokuchi et al., [10, 11], FSG by Kuramochi and Karypis, [12], Borgelt and Berthold, [1], gSpan by Yan and Han, [23] and FFSM by Huan et al., [9]. These algorithms aim to find the complete frequent pattern set, but suffer from the fact that due to combinatorial complexity, the size of the complete pattern set is exponential even for graphs of moderate sizes. To void the search on a complete pattern set, SPIN [17] and MARGIN [20] find all maximal patterns. Unfortunately, the number of all maximal patterns could still be too large to handle. ORIGAMI as proposed in [7] is an algorithm to find a representative pattern set based on output space sampling. Other works include structural leap search introduced by Yan et al. [22], which adopts structural similarity to mine significant graph patterns efficiently and directly from two graph datasets. Mining frequent pattern in single-graph setting is harder than mining in graph-transaction setting due to the complexity of support computation [13, 26]. One of the most recent work in this category is SpiderMine [26] which is designed to find the top-K largest patterns with a high probability of $1-\epsilon$ for any user-specified error bound ϵ. Due to its constraint on diameter bound and spider-based merging, it is hard for SpiderMine to find skinny patterns as we defined. Other important works in single graph setting include SUBDUE [8], SEuS was proposed by Ghazizadeh and Chawathe [6], and GREW [13], which have been thoroughly discussed in [26]. MoSS [5] is proposed for mining complete patterns in single graphs, which, as any other algorithm mining for the complete pattern set, suffers from the same scalability issue as the input graph size grows. All these algorithms are not designed to mine patterns with constraints directly as described in our direct mining framework.

Constraint-based pattern mining is of interest in a wide applications where a task integrates constraints into the mining process. Recent work has highlighted the importance of constraint-based mining in the context of mining frequent itemsets, sequential patterns, associations, and graphs. Pei et al. [16] identify a class of hard constraints called convertible constraints and develop its pushing method in itemset setting. The study [15] overviews the principles of pattern-growth methods for constrained frequent pattern

[2] http://www.weibo.com

mining according to the constraints in itemset settings and sequential pattern mining. Another work for itemset settings is proposed by Zhu et al. [28], which mines large patterns probabilistically by core pattern fusion. Ng et al. [14] study three important classes of constraints: monotonicity, antimonotonicity and succinctness and develops efficient constraint-based frequent itemset mining algorithms for a single constraint. Constraints in graph settings are more complicated [18]. The study gPrune [27] discusses pruning properties and techniques in both pattern space and data space. SkinnyMine, on the other hand, examines the properties of constraints that would admit our proposed direct mining framework.

Direct mining has been studied not on constraints on patterns but on certain utility of patterns such as discriminativeness measured on the mining result [4], which is a different problem from ours.

8. CONCLUSION

In this paper, we propose a direct mining framework to solve the problem of efficient constrained frequent graph mining and illustrate our ideas in the context of the problem of skinny pattern mining. Based on the key concept of a *canonical diameter*, we develop SkinnyMine, an efficient algorithm to mine all the l-long δ-skinny patterns guaranteeing both the completeness of our mining result as well as the unique generation of each target pattern. We also present a general direct mining framework together with two properties of *reducibility* and *continuity* for qualified constraints. We conducted extensive experiments on both synthetic and real data to demonstrate the effectiveness and efficiency of our approach.

9. ACKNOWLEDGMENTS

We thank Aoying Zhou and Weining Qian from East China Normal University for providing the Weibo data. This research is partially supported by Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office. Qiang Qu is supported by the Geocrowd Initial Training Network, funded by the European Commission as an FP7 Peoples Marie Curie Action under grant agreement number 264994.

10. REFERENCES

[1] C. Borgelt and M. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *ICDM*, pages 211–218, 2002.

[2] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. In *SIGMOD*, pages 857–872, 2007.

[3] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *TKDE*, 17(8):1036–1050, 2005.

[4] W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. S. Yu, and O. Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *SIGKDD*, pages 230–238, 2008.

[5] M. Fiedler and C. Borgelt. Support computation for mining frequent subgraphs in a single graph. In *The 5th International Workshop on Mining and Learning with Graphs*, 2007.

[6] S. Ghazizadeh and S. S. Chawathe. Seus: Structure extraction using summaries. In *ICDS*, pages 71–85, 2002.

[7] M. A. Hasan, V. Chaoji, S. Salem, J. Besson, and M. J. Zaki. Origami: Mining representative orthogonal graph patterns. In *ICDM*, pages 153–162, 2007.

[8] L. Holder, D. Cook, and S. Djoko. Substructure discovery in the subdue system. In *SIGKDD*, pages 169–180, 1994.

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

[10] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23, 2000.

[11] A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs:mining graph data. *Machine Learning*, 50(3):321–354, 2003.

[12] M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. In *TKDE*, pages 1038–1051, 2004.

[13] M. Kuramochi and G. Karypis. Grew-a scalable frequent subgraph discovery algorithm. In *ICDM*, pages 439–442, 2004.

[14] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Mining frequent itemsets with convertible constraints. In *SIGMOD*, pages 13–24, 1998.

[15] J. Pei and J. Han. Constrained frequent pattern mining: a pattern-growth view. In *SIGKDD*, pages 31–39, 2002.

[16] J. Pei, J. Han, and L. V. Lakshmanan. Mining frequent itemsets with convertible constraints. In *ICDE*, pages 433–442, 2001.

[17] J. Prins, J. Yang, J. Huan, and W. Wang. Spin: Mining maximal frequent subgraphs from graph databases. In *SIGKDD*, pages 581–586, 2004.

[18] Q. Qu, F. Zhu, X. Yan, J. Han, P. S. Yu, and H. Li. Efficient topological olap on information network. In *DASFAA*, pages 389–403, 2011.

[19] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. 4(11):992–1003, 2011.

[20] L. Thomas, S. Valluri, and K. Karlapalem. Margin: Maximal frequent subgraph mining. In *ICDM*, pages 1097–1101, 2006.

[21] H. Tong, B.A.Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *CIKM*, pages 245–254, 2012.

[22] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by leap search. In *SIGMOD*, pages 433–444, 2008.

[23] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

[24] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004.

[25] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: Tree + delta >= graph. In *VLDB*, pages 938–949, 2007.

[26] F. Zhu, Q. Qu, D. Lo, X. Yan, J. Han, and P. S. Yu. Mining top-k large structural patterns in a massive network. *PVLDB*, 4(11):807–818, 2011.

[27] F. Zhu, X. Yan, J. Han, and P. Yu. gprune: A constraint pushing framework for graph pattern mining. In *PAKDD*, pages 388–400, 2007.

[28] F. Zhu, X. Yan, J. Han, P. Yu, and H. Cheng. Mining colossal frequent patterns by core pattern fusion. In *ICDE*, pages 706–715, 2007.