gPrune: A Constraint Pushing Framework for Graph Pattern Mining

Feida Zhu[†] Xifeng Yan[†] Jiawei Han[†] Philip S. Yu[‡]
[†] Computer Science, UIUC, {feidazhu,xyan,hanj}@cs.uiuc.edu
[‡] IBM T. J. Watson Research Center, psyu@us.ibm.com

Abstract. In graph mining applications, there has been an increasingly strong urge for imposing user-specified constraints on the mining results. However, unlike most traditional itemset constraints, structural constraints, such as density and diameter of a graph, are very hard to be pushed deep into the mining process. In this paper, we give the first comprehensive study on the pruning properties of both traditional and structural constraints aiming to reduce not only the pattern search space but the data search space as well. A new general framework, called **gPrune**, is proposed to incorporate all the constraints in such a way that they recursively reinforce each other through the entire mining process. A new concept, *Pattern-inseparable Data-antimonotonicity*, is proposed to handle the structural constraints unique in the context of graph, which, combined with known pruning properties, provides a comprehensive and unified classification framework for structural constraints. The exploration of these antimonotonicities in the context of graph pattern mining is a significant extension to the known classification of constraints, and deepens our understanding of the pruning properties of structural graph constraints.

1 Introduction

Graphs are widely used to model complicated structures in many scientific and commercial applications. Frequent graphs, those occurring frequently in a collection of graphs, are especially useful in characterizing graph sets, detecting network motifs [2], discriminating different groups of graphs [3], classifying and clustering graphs [4–6], and building graph indices [7]. For example, Huan *et al.* [5] successfully applied the frequent graph mining technique to extract coherent structures and used them to identify the family to which a protein belongs. Yan *et al.* [7] chose discriminative patterns from frequent graphs and applied them as indexing features to achieve fast graph search.

Unfortunately, general-purpose graph mining algorithms cannot fully meet users' demands for mining patterns with their own constraints. For example, in computational biology, a highly connected subgraph could represent a set of genes within the same functional module [8]. In chem-informatics, scientists are often interested in frequent graphs that contain a specific functional fragment, *e.g.*, a benzine ring. In all these applications, it is critical for users to have control on certain properties of the mining results for them to be meaningful. However, previous studies have left open the problem of pushing sophisticated structural constraints to expedite the mining process. This gap between user demand and the capability of current known mining strategies calls for a constraint-based mining framework that incorporates these structural constraints.

Related Work. A number of efficient algorithms for frequent graph mining are available in data mining community, *e.g.*, AGM [14], FSG [15], the path-join algorithm [16], gSpan[17], MoFa[3], FFSM [18], SPIN[19] and Gaston [20]. Few of them considered the necessary changes of the mining framework if structural constraints are present. Constraint-based frequent pattern mining has been studied in the context of association rule mining by Ng *et al.* [9], which identifies three important classes of constraints: *monotonicity, antimonotonicity* and *succinctness* and develops efficient constraint-based frequent itemset mining algorithms for a single constraint. Ng *et al.* also pointed out the importance of exploratory mining of constrained association rules so that a user can be involved in the mining process. Other complicated constraints, such as gradients [26], block constraints [27], constraints on sequences [28], and connectivity constraints [23], are proposed for different applications. Pei *et al.* discovers another class of constraint *convertible constraints* and its pushing methods. Constrained pattern mining for graphs has been looked into by Wang *et al.*, [29], although only constraints with monotonicity/antimonotonicity/succinctness are discussed. Bucila *et al.* [10] introduced a DualMiner framework that simultaneously takes advantage of both monotonicity and antimonotonicity to increase mining efficiency. The general framework of these mining methods is to push constraints deep in order to prune pattern search space. Although this is effective in many cases, the greatest power of constraint-based frequent pattern mining is achieved only when considering together the reduction on both the pattern search space and the data search space. Bonchi *et al.*have taken successful steps in this direction by proposing ExAnte, [11–13], a pre-processor to achieve data reduction in constrained itemset mining. ExAnte overcame the difficulty of combining the pruning power of both anti-monotone and monotone constraints, the latter of which had been considered hard to exploit without compromising the anti-monotone constraints. Boulicaut and De Raedt [1] have explored constraint-based mining as a step towards inductive databases.

Our Contributions. In this paper we show that the data reduction technique can in fact be extended beyond the preprocessing stage as in ExAnte, and pushed deeper into the mining algorithm such that the data search space is shrunk recursively each time it is projected for a pattern newly grown, through the entire mining process. More importantly, our study of graph constraints shows that for sophisticated constraints in data sets whose structures are more complicated than itemsets, data space pruning could be effective only when the structural relationship between the embedded pattern and the data is taken into account. This new constraint property, which we term as *Pattern-inseparable D-antimonotonicity*, is unique in the context of graphs and, to our best knowledge, has not been explored before in literature. It distinguishes itself from other pruning properties in that most sophisticated structural constraints, *e.g.*, diameter, density, and connectivity, exhibit neither antimonotonicity nor monotonicity. Without exploiting Pattern-inseparable D-antimonotonicity, current mining algorithms would have to enumerate all frequent graphs in the first place and then check constraints on them one by one. The paper makes the following contributions: First, we present the first systematic study of the pruning properties for complicated structural constraints in graph mining which achieves pruning on both pattern and data spaces. The full spectrum of pruning power is covered by (1) extending the known antimonotonicities for itemsets to easier cases and (2) discovering novel pattern-separable and pattern-inseparable D-antimonotonicities to handle structural constraints when pattern embeddings have to be considered. Secondly, a general mining framework is proposed that incorporates these pruning properties in graph pattern mining. In particular, data space pruning is coupled tightly with other constraint-based pruning such that data reduction is exploited throughout the entire mining process. Thirdly, discussion is given on mining strategy selection when a trade-off has to be made between the naive enumerating-and-checking approach and our pruning-property-driven approach.

2 Preliminaries

As a convention, the *vertex set* of a graph P is denoted by $V(P)$ and the *edge set* by $E(P)$. For two graphs P and P', P is a subgraph of P' if there exists a subgraph isomorphism from P to P', denoted by $P \subseteq P'$. P' is called a supergraph of P. In graph mining, a pattern is itself a graph, and will also be denoted as P. Given a set of graphs $D = \{G_1, G_2, \ldots, G_n\}$, for any pattern P, the *support database of* P is denoted as D_P, and is defined as $D_P = \{G_i | P \subseteq G_i, 1 \leq i \leq n\}$. D_P is also referred to as the *data search space* of P, or *data space* for short. A graph pattern P is *frequent* if and only if $\frac{|D_P|}{|D|} \geq \sigma$ for a support threshold σ.

A constraint C is a boolean predicate on the pattern space U. Define $f_C : U \rightarrow \{0,1\}$ as the corresponding boolean function of C such that $f_C(P) = 1, P \in U$ if and only if P satisfies the constraint C. For example, let C be the constraint $Max\_Degree(P) \geq 10$ for a graph pattern P. Then $f_C(P) = 1$ if and only if the maximum degree of all the vertices of P is greater than 10. We formulate the constraint-based frequent graph pattern mining problem as the following:

Definition 1. (Constraint-based Frequent Graph Pattern Mining) *Given a set of graphs* $D = \{G_1, G_2, \ldots G_n\}$*, a support threshold* σ*, and a constraint* C*, constraint-based frequent graph pattern mining is to find all* P *such that* $\frac{|D_P|}{|D|} \geq \sigma$ *and* $f_C(P) = 1$.

Here are some graph constraints used in this paper: (1) The *density ratio* of a graph P, denoted as $Density\_Ratio(P)$, is defined as $Density\_Ratio(P) = \frac{|E(P)|}{|V(P)|(|V(P)|-1)/2}$. (2) The *Density* of a graph P is defined as $Density(P) = \frac{|E(P)|}{|V(P)|}$. (3) The *Diameter* of a graph P is the maximum length of the shortest path between any two vertices of P. (4) $EdgeConnectivity(P)(VertexConnectivity(P))$ is the minimum number of edges(vertices) whose deletion from P disconnects P.

3 Pattern Mining Framework

gPrune can be applied to both Apriori-based model and the pattern-growth model. In this paper, we take the pattern-growth model as an example to illustrate the pruning optimizations. Nevertheless, the techniques proposed here can also be applied to Apriori-based methods. The pattern-growth graph mining approach is composed of two stages (1) pattern seed generation (2) pattern growth. The mining process is conducted by iterating these two stages until all frequent patterns are found.

Pattern Seed Generation: We use gSpan[17] to enumerate all the seeds with size of increasing order. One pattern seed is generated every time and proceeds to the pattern growth stage. A pattern seed could be a vertex, an edge, or a small structure.

Pattern Growth: As outlined in Algorithm 1, PatternGrowth keeps a set S of pattern seeds and a set F of frequent patterns already mined. Each iteration of $PatternGrowth$ might generate new seeds (added to S), and identify new frequent patterns (added to

F). Line 1 initializes the data structures. Initially, S contains only the pattern seed. Line 2 to 11 grow every pattern seed in S until S is exhausted. For each pattern seed Q, which is taken from the set S in Line 3, Q is checked through its data search space and augmented incrementally by adding a new edge or vertex (Lines 4 and 5). Each augmented pattern is checked for pattern pruning in Line 6 and dropped whenever it satisfies the pattern pruning requirements. All the augmented patterns that survive the checking are recorded in S_t. Then in Line 8, for each surviving pattern, we construct its own support data space from that of Q's. Line 9 checks data pruning for each G in the support space. Since each thus augmented pattern is a frequent pattern, we add them to F in Line 10. Finally, these patterns are added to S, so that they will be used to grow new patterns. When S is exhausted, a new pattern seed will be generated until it is clear that all frequent patterns are discovered. The algorithm input: A frequent pattern seed P, graph database $D = \{G_1, G_2, \dots, G_n\}$ and the existing frequent pattern set F. The algorithm output: New frequent pattern set F.

Algorithm 1 PatternGrowth

```
1: $S \leftarrow \{P\}; F \leftarrow F \bigcup \{P\}; S_t \leftarrow \emptyset$
2: **while** $S \neq \emptyset$;
3:     $Q \leftarrow pop(S)$;
4:     **for each** graph $G \in D_Q$
5:         Augment $Q$ and save new patterns in $S_t$;
6:         $\boxed{\text{Check pattern pruning on each } P \in S_t;}$
7:     **for each** augmented pattern $Q' \in S_t$
8:         Construct support data space $D_{Q'}$ for $Q'$;
9:         $\boxed{\text{Check data pruning on } D_{Q'};}$
10:    $F \leftarrow F \bigcup S_t$;
11:    $S \leftarrow S \bigcup S_t$;
12: **return** $F$;
```

PatternGrowth checks for pattern pruning in Line 6 and data pruning in Line 9. Pattern pruning is performed whenever a new augmented pattern is generated. This means any unpromising pattern will be pruned before constructing its data search space. Notice that data pruning is performed whenever infrequent edges are dropped after a new data search space is constructed and offers chance to drop new target graphs. As such, the search space for a pattern keeps shrinking as the pattern grows.

4 Pruning Properties

A pruning property is a property of the constraint that helps prune either the pattern search space or the data search space. Pruning properties which enable us to prune patterns are called *P-antimonotonicity*, and those that enable us to prune data are called *D-antimonotonicity*.

4.1 Pruning Patterns

(1) Strong P-antimonotonicity

Definition 2. *A constraint C is* ***strong P-antimonotone*** *if $f_C(P') = 1 \rightarrow f_C(P) = 1$ for all $P \subseteq P'$.*

Strong P-antimonotonicity is simply the antimonotone property which has been known long since [9]. We call it strong P-antimonotonicity only to distinguish it from the other P-antimonotonicity introduced below. An example of strong P-antimonotone constraint for graph is acyclicity.

(2) Weak P-antimonotonicity

Constraints like "$Density\_Ratio(G) \geq 0.1$" is not strong P-antimonotone. Growing a graph G could make $Density\_Ratio(G)$ go either up or down. However, they have *weak P-antimonotonicity*, which is based on the following intuition. If a constraint C is not strong P-antimonotone, then there must exist a pattern P violating C and a supergraph of P, say Q, that satisfies C. In this case, we cannot prune graph P even if P violates C because Q might be missed if Q can only be grown out of P. However, if we can guarantee that Q can always be grown from some other subgraph P' such that P' satisfies C, we can then safely prune P.

Definition 3. *A constraint C is* ***weak P-antimonotone*** *if for a graph P' where $|V(P')| \geq k$ for some constant k, $f_C(P') = 1 \rightarrow f_C(P) = 1$ for some $P \subset P'$, such that $|V(P)| = |V(P')| - 1$.*

k is the size of the minimum instance to satisfy the constraint. When mining for weak P-antimonotone constraints, since we are sure that, for any constraint-satisfying pattern Q, there is a chain of substructures such that $g_1 \subset g_2 \subset ... \subset g_n = Q$ and g_i satisfies the constraint for all $1 \leq i \leq n$, we can drop a current pattern P if it violates the constraint, even if some supergraph of P might satisfy the constraint. Weak P-antimonotonicity allows us to prune patterns without compromising the completeness of the mining result. A similar property on itemsets, "loose antimonotonicity", has been discussed by Bonchi *et al.*in [13]. Notice that if a constraint is strong P-antimonotone, it is automatically weak P-antimonotone; but not vice versa. Also note that we can have similar definition of weak P-animonotonicity with the chain of substructure decreasing in number of edges.

We us the graph density ratio example to illustrate the pruning. The proof of the following theorem is omitted due to space limit.

Theorem 1. *Given a graph G, if $Density\_Ratio(G) > \delta$, then there exists a sequence of subgraphs $g_3, g_4, \ldots, g_n = G$, $|V(g_i)| = i$ $(3 \leq i \leq n)$ such that $g_3 \subset g_4 \subset \ldots g_n$ and $Density\_Ratio(g_i) > \delta$.*

Theorem 1 shows a densely connected graph can always be grown from a smaller densely connected graph with one vertex less. As shown in this example of graph density ratio, even for constraints that are not strong P-antimonotone, there is still pruning power to tap if weak P-antimonotonicity is available.

4.2 Pruning Data

(1) Pattern-separable D-antimonotonicity

Definition 4. *A constraint C is* ***pattern-separable D-antimonotone*** *if for a pattern P and a graph $G \in D_P$, $f_C(G) = 0 \rightarrow f_C(P') = 0$ for all $P \subseteq P' \subseteq G$.*

For constraints with pattern-separable D-antimonotonicity, the exact embeddings of the pattern are irrelevant. Therefore, we only need to check the constraint on the entire graphs in the pattern's data search space, and safely drop a graph if it fails the constraint.

Consider the constraint *"the number of edges in a pattern is greater than 10"*. The observation is that every time a new data search space is constructed for the current pattern P, we can scan the graphs in the support space and prune those with less than 11 edges.

It is important to recognize that this data reduction technique can be applied repeatedly in the entire mining process, instead of applying in an initial scan of the database as a preprocessing procedure. It is true that we will not benefit much if this data pruning is effective only once for the original data set, *i.e.*, if any graph surviving the initial scanning will always survive in the later pruning. The key is that ***in our framework, data pruning is checked on every graph in the data search space each time the space is updated for the current pattern. As such, a graph surviving the initial scan could still be pruned later.*** This is because when updating the search space for the current pattern P, edges which were frequent at last step could now become infrequent, and are thus dropped. This would potentially change each graph in the data search space, and offer chance to find new graphs with less than 11 edges which become eligible for pruning *only* at this step. Other examples of pattern-separable D-antimonotonic constraints include *path/feature containment*, *e.g.*, *pattern contains three benzol rings*.

(2) Pattern-inseparable D-antimonotonicity

Unfortunately, many constraints in practice are not pattern-separable D-antimonotone. $VertexConnectivity(P) > 10$ is a case in point. The exact embedding of the pattern is critical in deciding whether it is safe to drop a graph in the data search space. These constraints are thus pattern-inseparable. In these cases, if we "*put the pattern P back to G*", *i.e.*, considering P together with G, we may still be able to prune the data search space.

Definition 5. *A constraint C is* ***pattern-inseparable D-antimonotone*** *if for a pattern P and a graph $G \in D_P$, there exists a measure function $M : \{P\} \times \{G\} \rightarrow \{0, 1\}$ such that $M(P, G) = 0 \rightarrow f_C(P') = 0$ for all $P \subseteq P' \subseteq G$.*

The idea of using pattern-inseparable D-antimonotone constraints to prune data is the following. After embedding the current pattern P into each $G \in D_P$, we compute by a measure function, for all supergraphs P' such that $P \subset P' \subset G$, an upper/lower bound of the graph property to be computed in the constraint. This bound serves as a necessary condition for the existence of a constraint-satisfying supergragh P'. We discard G if this necessary condition is violated. For example, suppose the constraint is $VertexConnectivity(P) > 10$. If after embedding P in G, we find that the maximum vertex connectivity of all the supergraphs of P is smaller than 10, then no future pattern growing out of P in G will ever satisfy the constraint. As such G can be safely dropped. The measure function used to compute the bounds depends on the particular constraint. For some constraints, the computational cost might be prohibitively high and such a computation will not be performed. Another cost issue associated with pruning based on pattern-inseparable D-antimonotonicity is the maintenance of the pattern growth tree to track pattern embeddings. The Mining algorithm has to make a choice based on the cost of the pruning and the potential benefit. More discussion on the trade-off in these cases is given in Section 5. We use the vertex connectivity as an example to show how to perform data pruning. The time cost is linear in the pattern size for this constraint.

Let $Neighbor(P)$ be the set of vertices adjacent to pattern P. For the vertex connectivity constraint, the following lemma gives a necessary condition for the existence of a P' such that $VertexConnectivity(P') \geq \delta$.

Lemma 1. *If $|Neighbor(P)| < \delta$, then there exists no P' such that $P \subset P' \subset G$ and $VertexConnectivity(P') > \delta$.*

Therefore, for each pair of pattern P and $G \in D_P$, the measure function $M(P, G)$ could first embed P in G, and then identify $Neighbor(P)$. If $|Neighbor(P)|$ is smaller than 10, returns 0. This pruning check is computationally cheap and only takes time linear in $|V(G - P)|$.

Constraint	strong P-antimonotone	weak P-antimonotone	pattern-separable D-antimonotone	pattern-inseparable D-antimonotone
$Min\_Degree(G) \geq \delta$	No	No	No	Yes
$Min\_Degree(G) \leq \delta$	No	Yes	No	Yes
$Max\_Degree(G) \geq \delta$	No	No	Yes	Yes
$Max\_Degree(G) \leq \delta$	Yes	Yes	No	Yes
$Density\_Ratio(G) \geq \delta$	No	Yes	No	Yes
$Density\_Ratio(G) \leq \delta$	No	Yes	No	Yes
$Density(G) \geq \delta$	No	No	No	Yes
$Density(G) \leq \delta$	No	Yes	No	Yes
$Size(G) \geq \delta$	No	Yes	Yes	Yes
$Size(G) \leq \delta$	Yes	Yes	No	Yes
$Diameter(G) \geq \delta$	No	Yes	No	Yes
$Diameter(G) \leq \delta$	No	No	No	Yes
$EdgeConnectivity(G) \geq \delta$	No	No	No	Yes
$EdgeConnectivity(G) \leq \delta$	No	Yes	No	Yes
G contains P (e.g., P is a benzol ring)	No	Yes	Yes	Yes
G does not contain P (e.g., P is a benzol ring)	Yes	Yes	No	Yes

Fig. 1. A General Picture of Pruning Properties of Graph Constraints

We summarize our study on the most useful constraints for graphs in Figure 1. Proofs are omitted due to space limit.

5 Mining Strategy Selection

The checking steps for pruning patterns and data are both associated with a computational cost. Alternatively, one can first mine all frequent patterns by a known mining algorithm, gSpan [17] for example, then check constraints on every frequent pattern output, and discard those that do not satisfy the constraint. We call this method the enumerate-and-check approach in the following discussion. Which approach is better depends on the total mining cost in each case. The best strategy therefore is to estimate the cost and potential benefit for each approach at every pruning step, and adopt the one that would give better expected efficiency.

The growth of a pattern forms a partial order $\prec$ defined by subgraph containment, *i.e.*, $P \prec Q$ if and only if $P \subseteq Q$. The partial order can be represented by a pattern tree model in which a node P is an ancestor of a node Q if and only if $P \prec Q$.

Each internal node represents a frequent pattern, which is associated with its own data search space $\{T_i\}$. The execution of a pattern mining algorithm can be viewed as growing such a pattern tree. Every initial pattern seed is the root of a new tree. Every time it augments a frequent pattern P, it generates all P's children in the tree. As such, each leaf corresponds to an infrequent pattern, or, in our mining model, a pattern that does not satisfy the constraints, since it is not further grown. Accordingly, the running time of a pattern mining algorithm can be bounded by the total number of nodes it generates in such a tree. This total sum is composed of two parts: (1) the set of all internal nodes, which corresponds to all the frequent patterns and is denoted as F; and (2) the set of all leaves, denoted by L, which corresponds to the infrequent patterns or constraint-violating patterns.

Let's look at the running time of the enumerate-and-check approach. Let the minimum support threshold be σ, *i.e.*a frequent pattern has to appear in at least $\sigma|D|$ graphs, where D is the entire graph database. Let $T_c(P)$ be the cost to check a constraint C on a graph P. The running time of the enumerate-and-check approach can be lower-bounded as follows:

1. Internal Nodes
   If an augmented pattern P is frequent, at least $\sigma|D|$ time has to be spent in the frequency checking and data search space construction. Hence, the construction time for such a node P is $|D_P| + T_c(P) \geq \sigma|D| + T_c(P)$.
2. Leaf Nodes
   If P is infrequent, at least $\sigma|D|$ time has to be spent in frequency checking. Since frequency checking is limited to support data space of P's parent node, denoted as $Parent(P)$, the construction time for P is $\geq min(\sigma|D|,|D_{Parent(P)}|)= \sigma|D|$.

Then the total cost, T_P, for mining from an initial pattern seed P by the enumerate-and-check approach is lower-bounded as $T_P \geq \sum_{P_i \in F_P}(\sigma|D|+T_c(P_i))+\sum_{P_i \in L}\sigma|D|$ $=\sigma|D||F_P|+\sum_{P_i \in F_P} T_c(P_i)+\sigma|D||L| \geq 2\sigma|D||F_P|+\sum_{P_i \in F_P} T_c(P_i)$, where F_P is the set of frequent patterns grown from P. Essentially, the time cost of the enumerate-and-check approach is proportional to $|F_P|$. To bound $|F_P|$ means to bound the number of frequent patterns that would be generated from a pattern P. It is very hard to analytically give an accurate estimation of this heavily-data-dependent quantity. Our empirical studies show that in general, $|F_P|$ is very large for small patterns. An upper-bound of $|F_P|$ can also be proved to be $\Theta(2^{|G|}), G \in D_P$. The proof is omitted due to space constraint.

Now we show how we should choose between the enumerate-and-check approach and our mining framework in both pattern pruning and data pruning cases.

1. **Pruning Patterns**: If we can prune a frequent pattern P after checking constraints on it, then the entire subtree rooted at P in the pattern tree model will not be grown. The time we would save is T_P. The extra time spent for constraint checking is $T_c(P)$. If it turns out that we can not prune it after the checking, we will grow it as in the enumerate-and-check approach. The expected cost of using our mining model is $T_{prune} = T_c(P)+(1-p)\cdot T_P$ where p is the probability that P fails the constraint checking. $T_c(P)$ depends on the algorithm used for the constraint checking, while p will be estimated empirically. The cost of the enumerate-and-check approach is T_P. As such, pattern pruning should be performed when $T_c(P) \leq p \cdot T_P$.
2. **Pruning Data**: If we can prune a graph G from the data search space of P after data pruning checking, G will be pruned from the data search spaces of all nodes in the subtree rooted at P. Therefore the time we save is lower-bounded by T_P. Let $T_d(P,G)$ be the time cost to check data pruning for a pattern P and a graph $G \in D_P$. Let q be the probability that G can be discarded after checking data pruning. Then for a graph $G \in D_P$, using data pruning by our model takes expected time $T_{prune} = T_d(P,G) + (1-q)T_P$, while the enumerate-and-check approach would cost time T_P. The probability q can be obtained by applying sampling technique on D_P. We would perform data pruning checking for G if $T_d(P,G) < q \cdot T_P$. Otherwise, we shall just leave G in the search space.

6 Experimental Evaluation

In this section, we are going to demonstrate the pruning power provided by the the new antimonotonicities introduced in our framework, *i.e.*, weak pattern-antimonotonicity and data-antimonotonicity. Among all of structural constraints described in Figure 1, minimum density ratio and minimum degree are selected as representatives. All of our experiments are performed on a 3.2GHZ, 1GB-memory, Intel PC running Windows XP.

We explored a series of synthetic datasets and two real datasets. The synthetic data generator[1] is provided by Yan *et al.*[23], which includes a set of parameters that allow a user to test the performance under different conditions. There are a set of parameters for users to specify: the number of target 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 each target graph (T), the average density of seed graphs (D), and the average density of noise edges in target graphs.

The detailed description about this synthetic data generator is referred to [23]. For a dataset which has 60 relational graphs of 1,000 distinct objects, 20 seed graphs (each seed graph has 10 vertices and an average density 0.5), 10 seed graphs per relational graph, and 20 noise edges per object ($0.01 \times 1,000 \times 2$), we represent it as N60O1kS20T10I10 D0.5d0.01.

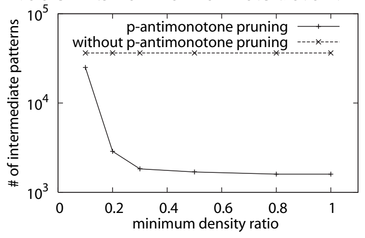

Fig. 2. Weak P-Antimonotonicity

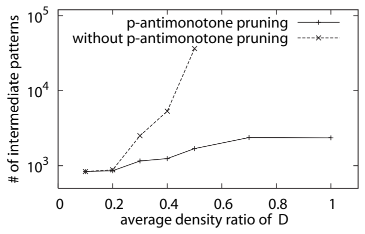

Fig. 3. Weak P-Antimonotonicity

The first experiment is about the minimum density ratio constraint. As proved in Section 4, minimum density ratio has weak pattern-antimonotonicity property. For each graph whose density ratio is greater than δ ($0 < \delta \leq 1.0$), we can find a subgraph with one vertex less whose density is greater than δ. That means we can stop growing any frequent pattern with one more vertex if its density is less than δ.

Figure 2 shows the pruning performance with various minimum density ratios. The data set used here is N60O1kS20T10I10 D0.5d0.01. The Y axis depicts the intermediate frequent patterns that are accessed during the mining process. The fewer the intermediate patterns, the better the performance, given the cost of checking the pattern's density ratio is negligible. The two curves show the performance comparison between methods with and without weak P-antimonotone pruning. As the figure shows, with the integration of the minimum density ratio constraint, we only need to examine much fewer frequent patterns, which proves the effectiveness of weak pattern-antimonotonicity. In the next experiment, we fix the density ratio threshold at 0.5 and change the average density ratio of seed graphs (D) in the above synthetic dataset. The denser the seed graphs, the more the dense subgraph patterns. It could take longer to find these patterns. Figure 3 depicts the performance comparison between methods with or without weak

[1] It will produce a distinctive label for each node in a graph

P-antimonotone pruning. We found that when D is greater than 0.6, the program without P-antimonotone pruning cannot finish in hours, while the one with P-antimonotone pruning can finish in 200 seconds.

Besides the weak P-antimonotone pruning, we also examined pattern inseparable D-antimonotonicity to pruning the data search space for the density ratio constraint. Given a frequent subgraph P and a graph G in the database ($P \subseteq G$), we need a measure to quickly check the maximum density ratio for each graph Q, where $P \subseteq Q \subseteq G$. For this purpose, we developed an algorithm for fast maximum density ratio checking. Let P' be the image of P in G. Our algorithm has three steps: (1) transform G to G' by merging all of the nodes in P'. (2) apply Goldberg's maximum density ratio subgraph finding algorithm to find a maximum density ratio subgraph in G' (time complexity $O(n^3 logn)$, where $n = |V(G')|$) [24]. (3) for graph G, calculate a maximum density ratio subgraph that contains P'; if this density ratio is below the density ratio threshold, we can safely drop G from the data search space of P(*i.e.*, G does not contain any subgraph Q that contains P and whose density ratio is greater than the threshold). For each discovered subgraph, we perform this checking to prune the data search space as much as possible. Although this checking is much faster than enumerating all subgraphs in G, we find it runs slower than a method without pruning, due to the high computational cost. That is, the cost model discussed in Section 5 does not favor this approach. Through this exercise, it was learned that, in order to deploy D-antimonotone pruning, the corresponding measure function in Defintion 5 has to be fast enough.

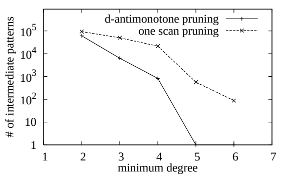

Fig. 4. P-Inseparable D-Antimonotonicity's effect on reducing # of intermediate patterns

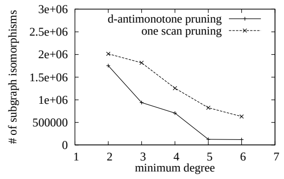

Fig. 5. P-Inseparable D-Antimonotonicity's effect on reducing # of subgraph isomorphism testings

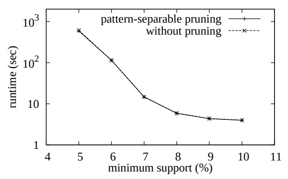

Fig. 6. P-Separable D-Antimonotonicity's effect on runtime

We applied the above frequent dense subgraph mining algorithm to the real data that 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. 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. The patterns mined by our methods exhibit strong biological meanings. The mining result was published in our previous work [23].

Although we did not find a good D-antimonotonicity for the density ratio constraint, D-antimonotonicity is still applicable for other constraints, e.g., the minimum degree constraint. Neither pattern antimonotonicity nor weak pattern antimonotonicity is available for the minimum degree constraint. Thus, we develop a pruning technique using pattern-inseparable data antimonotonicity, which checks the minimum degree of a pattern embedded in each graph. If the degree is below threshold δ, we drop the corresponding graph from the data search space. The dropping will also decrease the frequency of each pattern and its superpatterns, which may make them infrequent as a result.

Figure 4 shows the comparison of pruning performance between data-antimonotonicity and a one-scan pruning method that drops vertices with less than δ edges before running PatternGrowth. When the minimum degree constraint is weak, *e.g.*, minimum degree threshold is low, these two methods have similar performance. However, when the constraint becomes strong, the pruning based on data-antimonotonicity performs much better.

Figure 5 shows the number of subgraph isomorphisms performed for these two algorithms. It is clear that, using data antimonotonicity, a lot of graphs are pruned in the early stage so that the number of subgraph isomorphisms done in the later stage can be significantly reduced. We now check one constraint with pattern separable D-antimonotonicity — the minimum size constraint. The minimum size constraint on frequent itemset mining and sequential pattern mining has been explored before, e.g., SLPMiner developed by Seno and Karypis [25]. Suppose our task is to find frequent graph patterns with minimum length δ. One approach is to check the graphs in the data search space of each discovered pattern P and prune the graphs that are not going to generate patterns whose size is no less than δ. We developed several heuristics and applied our algorithm to mine the AIDS antiviral screen compound dataset from Developmental Theroapeutics Program in NCI/NIH [30]. The dataset contains 423 chemical compounds that are proved active to HIV virus.

Figure 6 shows the runtime of the two algorithms with and without pattern separable D-antimonotone pruning, with different support thresholds. The size constraint is set in a way such that less than 10 largest patterns are output. It is a surprise that for this dataset, pattern separable D-antimonotone pruning is not effective at all. Closer examination of this dataset reveals that most of the graphs can not be pruned because the sizes of frequent patterns are relatively small in comparison with the graphs in the database. This once again demonstrates, as also shown in the density ratio constraint, that the effectiveness of the integration of a constraint with the mining process is affected by many factors, e.g., the dataset and the pruning cost.

7 Conclusions

In this paper, we investigated the problem of incorporating sophisticated structural constraints in mining frequent graph patterns over a collection of graphs. We studied the nature of search space pruning for both patterns and data, and discovered novel antimonotonicities that can significantly boost pruning power for graph mining in each case: (1) weak pattern-antimonotonicity for patterns; (2) pattern-separable and pattern-inseparable data-antimonotonicities for data. We showed how these properties can be exploited to prune potentially enormous search space. An analysis of the trade-off between the enumerating-and-checking approach and the antimonotonicity-based approach was also given in this study.

References

1. Boulicaut,J.,De Raedt, L.: Inductive Databases and Constraint-Based Mining. (ECML'02) Tutorial.
2. Koyuturk, M., Grama, A., Szpankowski, W.: An efficient algorithm for detecting frequent subgraphs in biological networks. (ISMB'04). 200–207

3. Borgelt, C., Berthold, M.R.: Mining molecular fragments: Finding relevant substructures of molecules. (ICDM'02), 211–218
4. Deshpande, M., Kuramochi, M., Karypis, G.: Frequent sub-structure-based approaches for classifying chemical compounds. (ICDM'03). 35–42
5. Huan, J., Wang, W., Bandyopadhyay, D., Snoeyink, J., Prins, J., Tropsha, A.: Mining spatial motifs from protein structure graphs. (RECOMB '04), 308–315
6. Deshpande, M., Kuramochi, M., Wale, N., Karypis, G.: Frequent substructure-based approaches for classifying chemical compounds. IEEE TKDE **17**(8) (2005) 1036–1050
7. Yan, X., Yu, P.S., Han, J.: Graph indexing: A frequent structure-based approach.(SIGMOD'04), 335–346
8. Butte, A., Tamayo, P., Slonim, D., Golub, T., Kohane, I.: Discovering functional relationships between rna expression and chemotherapeutic susceptibility. In: Proc. of the National Academy of Science. Volume 97. (2000) 12182–12186
9. Ng, R., Lakshmanan, L.V.S., Han, J., Pang, A.: Exploratory mining and pruning optimizations of constrained associations rules. (SIGMOD'98), 13–24
10. Bucila, C., Gehrke, J., Kifer, D., White, W.: DualMiner: A dual-pruning algorithm for itemsets with constraints. Data Mining and Knowledge Discovery **7** (2003) 241–272
11. Bonchi, F., Giannotti, F., Mazzanti, A., Pedreschi, D.: Exante: Anticipated data reduction in constrained pattern mining. (PKDD'03)
12. Bonchi, F., Giannotti, F., Mazzanti, A., Pedreschi, D.: Exante: A preprocessing method for frequent-pattern mining. In: IEEE Intelligent Systems 20(3). (2005) 25–31
13. Bonchi, F., Lucchese, C.: Pushing tougher constraints in frequent pattern mining. (PAKDD'04), 114 – 124
14. Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. (PKDD'00), 13–23
15. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. (ICDM'01), 313–320
16. Vanetik, N., Gudes, E., Shimony, S.E.: Computing frequent graph patterns from semistructured data. (ICDM'02), 458–465
17. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. (ICDM'02), 721–724
18. Huan, J., Wang, W., Prins, J.: Efficient mining of frequent subgraph in the presence of isomorphism. (ICDM'03), 549–552
19. Prins, J., Yang, J., Huan, J., Wang, W.: Spin: Mining maximal frequent subgraphs from graph databases. (KDD'04), 581–586
20. Nijssen, S., Kok, J.: A quickstart in frequent structure mining can make a difference. (KDD'04), 647–652
21. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. (VLDB'94), 487–499
22. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. (SIGMOD'00), 1–12
23. Yan, X., Zhou, X.J., Han, J.: Mining closed relational graphs with connectivity constraints. (KDD'05), 324–333
24. Goldberg, A.: Finding a maximum density subgraph. Berkeley Tech Report, CSD-84-171
25. Seno, M., Karypis, G.: Slpminer: An algorithm for finding frequent sequential patterns using length decreasing support constraint. (ICDM'02), 418–425
26. Dong, G., Han, J., Lam, J., Pei, J., Wang, K., Zou, W.: Mining constrained gradients in multi-dimensional databases. IEEE TKDE **16** (2004) 922–938
27. Gade, K., Wang, J., Karypis, G.: Efficient closed pattern mining in the presence of tough block constraints. (KDD'04), 138 – 147
28. Zaki, M.: Generating non-redundant association rules.(KDD'00), 34–43
29. Wang, C., Zhu, Y., Wu, T., Wang, W., Shi, B.: Constraint-based graph mining in large database. (In: APWeb 2005) 133 – 144
30. Yan, X., Han, J.: CloseGraph: Mining Closed Frequent Graph Patterns (KDD'03)286-295