Direct Mining of Discriminative and Essential Frequent Patterns via Model-based Search Tree

Wei Fan[†], Kun Zhang[‡], Hong Cheng[*], Jing Gao[*],
Xifeng Yan[†], Jiawei Han[*], Philip S. Yu[#], and Olivier Verscheure[†]
[†]IBM T.J.Watson Research Center
[‡]Xavier University of Louisiana
[*]University of Illinois at Urbana Champaign
[#]University of Illinois at Chicago
{weifan,xifeng,ov1}@us.ibm.com, kzhang@xula.edu,
{hcheng3,jgao3,hanj}@uiuc.edu, psyu@cs.uic.edu

ABSTRACT

Frequent patterns provide solutions to datasets that do not have well-structured feature vectors. However, frequent pattern mining is non-trivial since the number of unique patterns is exponential but many are non-discriminative and correlated. Currently, frequent pattern mining is performed in two sequential steps: enumerating a set of frequent patterns, followed by feature selection. Although many methods have been proposed in the past few years on how to perform each separate step efficiently, there is still limited success in eventually finding highly compact and discriminative patterns. The culprit is due to the inherent nature of this widely adopted two-step approach. This paper discusses these problems and proposes a new and different method. It builds a decision tree that partitions the data onto different nodes. Then at each node, it directly discovers a discriminative pattern to further divide its examples into purer subsets. Since the number of examples towards leaf level is relatively small, the new approach is able to examine patterns with extremely low global support that could not be enumerated on the whole dataset by the two-step method. The discovered feature vectors are more accurate on some of the most difficult graph as well as frequent itemset problems than most recently proposed algorithms but the total size is typically 50% or more smaller. Importantly, the minimum support of some discriminative patterns can be extremely low (e.g. 0.03%). In order to enumerate these low support patterns, state-of-the-art frequent pattern algorithm either cannot finish due to huge memory consumption or have to enumerate 10^1 to 10^3 times more patterns before they can even be found. Software and datasets are available by contacting the author.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications-Data Mining

General Terms

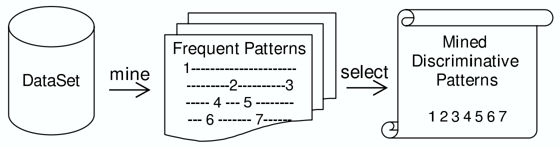
Algorithms

1. INTRODUCTION

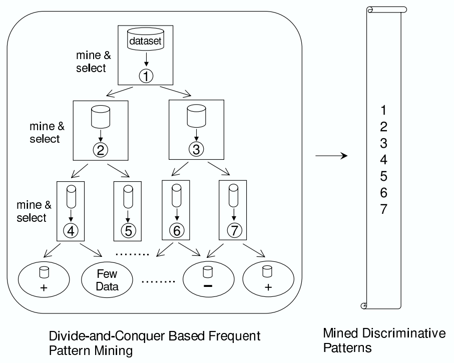
Many real-world data mining problems have no pre-defined feature vectors that can be given to data mining algorithms to construct predictive models. Facing these challenges, frequent-patterns (e.g., frequent itemsets [3, 13], graph mining [17, 25] and sequential pattern mining [4, 21], etc) have been proposed and actively studied as candidate feature sets. These methods look for statistically significant structures hidden in "raw" data. The main challenge and research interest for frequent pattern mining is how to discover those discriminative and essential patterns efficiently. It has been proved that frequent-pattern enumeration is NP-complete [26]. For some graph mining problems, when the relative support is 5%, the number of mined closed subgraphs (obvious redundancy excluded), can be up to $\times 10^7$. However, importantly, most discovered patterns either do not carry much information gain or are correlated in their predictability. On the other hand, if the support is set too low (such as $< 3\%$), the program simply may not finish since the virtual memory can be exhausted.

State-of-the-art frequent pattern mining algorithms [9, 5] employ a batch process, which first enumerates features above the given support and then performs feature selection on this initial pool, as shown in Figure 1 (a). Research interests in this area focus on increasing the effectiveness of each of these steps, for example, on how to efficiently enumerate those unique and non-overlapping patterns, how to prune the search tree to avoid enumerating patterns that are unlikely to carry much information gain, how to ensure that each example is covered by sufficient number of patterns, etc. These methods are important improvements, however, there is still limited success in eventually finding those small set of discriminative features, and this is due to inherent problems of the batch process as discussed below.

First, the number of candidate patterns can still be too large ($\approx \times 10^6$) for effective feature selection at the second



(a) Two Step Batch Approach


dataset
mine & select
1
2
3
4
5
6
7
Few Data
+
−
Divide-and-Conquer Based Frequent Pattern Mining
Mined Discriminative Patterns

(b) Divide-Conquer Approach

Figure 1: Batch vs. Divide-Conquer

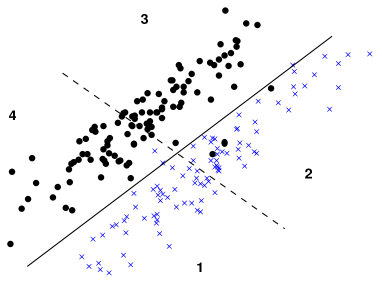
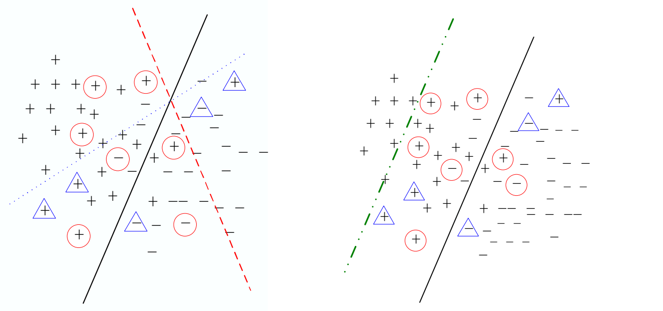


Figure 2: Information Gain in Different Subspace



(a) Uncorrelated patterns and their decision boundaries

(b) Decision boundary when both patterns considered

Figure 3: Uncorrelated Patterns ≠ higher accuracy

step. Second, if the frequency of discriminative features is below the support value chosen to enumerate the candidates, those features won't even be considered. Third, the discriminative power of each pattern is directly evaluated against the complete dataset, but not on some subset of examples that the other chosen patterns do not predict well. As demonstrated by a synthetic example using Gaussian mixture model in Figure 2, a feature not quite predictive on the complete dataset, can actually be quite informative on a subspace. On the complete data space, the solid line has high information gain since it can almost clearly separate the two classes of data, while the dashed line is not quite useful. However, after the solid line separates the two classes, the dashed line can make the divided subspaces "purer", such that both region 1 and 3 contain only one class of examples. In the batch mode, patterns like the "dashed line" is unlikely to be chosen by feature selection since the criteria usually operates on the complete dataset. Fourth, the correlation among multiple features are not directly evaluated on their joint predictability. Assume that a feature which can correctly predict some examples is already selected, to improve the overall accuracy, the features considered subsequently need to predict better on those examples or subspaces of the dataset that the chosen feature cannot predict correctly. However, the batch approach does not address this directly, but prefers features that are uncorrelated, either using covariance-based or coverage-based criteria. When two features are uncorrelated, they may not necessarily help each other to cover examples that each of them does not predict well by itself. Figure 3 (a) shows a dataset with two classes and two patterns α and β occurring in the data. Red circle represents the occurrence of α in certain examples and blue triangle represents the occurrence of β in some others. The red dashed line is the decision boundary which separates the examples based on whether they contain pattern α or not, while the blue dotted line is the decision boundary separating the examples based on the occurrences of pattern β. The optimal boundary is represented by the solid line which could not be derived solely based on α or β. Figure 3 (b) shows the decision boundary (in dotted dashed line) when both patterns are considered simultaneously. Although α and β are uncorrelated as they appear in disjoint examples, the selection of both at the same step will not necessarily help improve the accuracy. On the other hand, given that one feature or a number of features is already chosen, the next ideal feature should be the one that can cover the subset of examples where currently mined patterns do not cover well. Importantly, this feature can be correlated with chosen features on all other examples. The bottom line is that correlation criteria may not be a direct heuristic to look for predictive features. One should purposefully look for features that can correctly cover subspaces where currently discovered features cannot.

To solve the inherent problems of the batch process and find small set of highly discriminative patterns, we propose a divide-and-conquer based approach to directly mine discriminative patterns as features vectors. As shown in Figure 1 (b), the basic flow of the proposed algorithm proceeds by constructing a "model-based" search tree. The concept of this search tree is quite different from traditional frequent pattern-based search tree where each node normally denotes a sub-item and a path in the tree is a frequent pattern. In the model-based search tree, each node maintains a discovered pattern as the discriminative feature. Examples in the original dataset are sorted and partitioned down the tree. As each node being expanded, a frequent-pattern algorithm is invoked only on the examples that the node is responsible of. The frequent pattern with the highest information gain is chosen as the feature and maintained at the current node. Then based on the containment rule, the examples at the given node are partitioned into two disjoint subsets. The search and tree construction terminates when 1) either every example in the given node belongs to the same class or 2) the number of examples is less than a given threshold.

2. MODEL-BASED SEARCH TREE

Algorithm 1 presents the recursive method that builds the model-based search tree. The basic idea is to partition the data in a top-down manner and construct the tree using the best feature at each step. It starts with the whole data set and mines a set of frequent patterns from the data. The best pattern is selected according to some criterion and used to divide the data set into two subsets, one containing this pattern and the other not. The mining and pattern selection procedure is repeated on each of the subsets until the subset is small enough or the examples in the subset have the same class label. After the algorithm completes, a small set of informative features are uncovered and the corresponding model-based search tree is constructed.

Algorithm 1 Build Model-based Search Tree

```
Input: 1: A set of examples $\mathcal{D}$ from which
          features are to be mined
       2: A support threshold $p$ normalized
          between 0 and 1
       3: A feature discover algorithm, such as
          a frequent pattern algorithm $fp(\ldots)$.
       4: $m$ minimum node size.
Output: 1: A selected set of features, $\mathcal{F}_s$
        2: A model-based search tree $T$.

1: Call the frequent pattern algorithm, which returns
     a set of frequent patterns $\mathcal{FP} = fp(\mathcal{D}, p)$;
2: Evaluate the fitness of each pattern $\alpha \in \mathcal{FP}$;
3: Choose the best pattern $\alpha_m$ as the feature;
4: $\mathcal{F}_s = \mathcal{F}_s \cup \{\alpha_m\}$;
5: Maintain pattern $\alpha_m$ as the testing feature
     in current node of the tree $T$;
5: $\mathcal{D}_L$ = subset of examples in $\mathcal{D}$ containing $\alpha_m$;
6: $\mathcal{D}_R = \mathcal{D} - \mathcal{D}_L$;
7: for $\ell \in \{L, R\}$
8:     if $|\mathcal{D}_\ell| \leq m$ or examples in $\mathcal{D}_\ell$ have the same class
9:         label, make $T_\ell$ a leaf node;
10:    else
11:        recursively construct $T_\ell$ with $\mathcal{D}_\ell$ and $p$;
12: return $\mathcal{F}_s$ and $T$
```

2.1 Pattern Enumeration Scalability Analysis

The scale of patterns returned by frequent pattern algorithm can be formulated by $O\big((c_1 \cdot s)^{c_2 \cdot s(1-p)}\big)$, where s is scaled size of the dataset (> 3) and p is the support in percentage. The two constants c_1 and c_2 depend on both the dataset and the particular frequent pattern algorithm, and can be factored out. Thus, for simplicity, the scale of the problem is approximately $O\big(s^{s(1-p)}\big)$. Clearly, when p is set too low and the dataset is big, the number of patterns can be explosively large. Next, we look at how the proposed divide and conquer-based approach can significantly reduce the scale of the problem.

In the divide-conquer algorithm, without loss of generality, let us assume equal split. In fact, according to complexity analysis, unequal split will have the same big O result as that of the equal split and the difference is only in the constants. Then the number of patterns mined at each node of the tree (both leaf and non-leaf) can be expressed by the following recursive function:

$$T(s) = s^{s(1-p)} + 2T(s/2) \tag{1}$$

The upper bound of the number of frequent patterns ever enumerated and considered by the recursive method is shown in the following theorem.

Theorem 1. *For a problem of size s and support p, the recursive algorithm enumerates $O(s^{s(1-p)})$ number of patterns in total during the tree construction process.*

Proof. For a general recurrence problem: $T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function, the Master Theorem provides the solution to such problems. Specifically, in one of the three cases, if $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

For our problem, we show that it satisfies the conditions. First, $s^{s(1-p)}$ is an asymptotically positive function and $a, b = 2$ are both positive integers. Second, the problem has limited size, so s is bounded by an integer M. Let $\epsilon = M(1-p)-1$, which is greater than 0, then it is evident that $s^{s(1-p)} = \Omega(s^{log_2 2+\epsilon})$. Finally, to prove that $2(s/2)^{(1-p)s/2} \leq c(s^{s(1-p)})$, we need to show that

$$4\frac{s^{s(1-p)}}{2^{s(1-p)}} \leq cs^{s(1-p)}s^{s(1-p)}$$

which is equivalent to

$$(2s)^{s(1-p)} \geq 4/c$$

Let $c = 1/2$, the above inequality is true since $s > 3$ and p is usually a small number so that $s(1-p) \geq 3/2$, thus $(2s)^{s(1-p)} \geq 8$. Since this case applies to our problem, we could make the conclusion that the recursive algorithm considers $O(s^{s(1-p)})$ number of patterns. ☐

It is worth noting that in the recursive algorithm, the support p is the support at each node, i.e., support is calculated among all records falling into the node. Actually, a pattern with support p at a node will have a global support p', which is much smaller than p. For example, assume that the leaf node size is 10 and $p = 20\%$. For a problem size $n = 10000$, the normalized support in the complete dataset is $10 \times 20\%/10000 = 0.02\%$.

To find such patterns, the traditional pattern mining algorithms will return an explosive number of patterns or fail due to resource constraints, since it will generate $O(s^{s(1-p')})$ patterns, which is a huge number. Suppose p' is close to 0, then $1 - p' \simeq 1$ and pattern mining algorithms could obtain up to s^s patterns. However, the recursive algorithm could identify such patterns without considering every pattern, thus will not generate explosive number of patterns.

According to Theorem 1, the upper bound of the number of patterns is $s^{s(1-p)}$. Comparing with traditional pattern mining approaches, the "scale down" ratio of the pattern numbers will be up to

$$\simeq s^{s(1-p)}/s^s = \frac{1}{s^{sp}} \tag{2}$$

This demonstrates that the proposed recursive algorithm could get over the barrier of explosive growth of frequent patterns and successfully identify discriminative patterns with very small support.

2.2 Bound on Number of Returned Features

Now we consider a different problem, the upper bound on the number of discriminative features returned by the recursive method. In the worst case, the tree is complete and every leaf node has exactly m examples, thus the upper bound is

$$O(2^{\log_m(s)-1} - 1) < O(s/2 - 1) = O(s) = O(n) \tag{3}$$

since $m > 2$ and scaled problem size is exactly the number of examples n for the model-based search tree.

2.3 Subspace Pattern Selection

Assume we have a two-class problem with $\mathcal{C} = \{0, 1\}$ and a pattern α. Examples with class label 1 are called positive examples and examples from class 0 are negatives. Let C_0 and C_1 be the number of negative and positive examples respectively; P_0 and P_1 be the number of occurrences of the pattern α among negative and positive class examples respectively. Let x denote an example from either negative or positive class, then by definition, the information gain of the pattern α is measured as

$$\begin{aligned} & IG(\mathcal{C}|X) = H(\mathcal{C}) - H(\mathcal{C}|X) \\ & = -\sum_{c\in\{0,1\}} P(c)\log P(c) + \sum_{x\in\{0,1\}} P(x) \sum_{c\in\{0,1\}} P(c|x)\log P(c|x) \end{aligned}$$

where $P(c=1|x=1) = \frac{P_1}{P_0+P_1}$, $P(c=0|x=1) = \frac{P_0}{P_0+P_1}$, $P(c=1|x=0) = \frac{C_1-P_1}{C_1+C_0-P_1-P_0}$, $P(c=1|x=1) = \frac{C_0-P_0}{C_1+C_0-P_1-P_0}$. The key components in information gain definition are the following two proportion, $\frac{P_0}{C_0}$ and $\frac{P_1}{C_1}$: the more difference between these two terms, the higher is the information gain.

Now consider only a subset of examples are selected from the original set. Further, assume C'_0 and C'_1 are the number of negative and positive examples respectively; P'_0 and P'_1 be the number of occurrences of α in negative and positive class respectively, thus $C'_i < C_i$ and $P'_i < P_i$, $i \in \{0, 1\}$.

In the original data set, if the relative frequency of α in the positive and negative classes is very close, i.e., $\frac{P_0}{C_0} \sim \frac{P_1}{C_1}$, then α is not discriminative and the information gain of α is low, and should not be chosen. However, in the data subset with some examples eliminated, the relative frequency of α in the positive and negative class could be different, thus making $\frac{P_0}{C_0} \ll \frac{P_1}{C_1}$ or vice versa. In such cases, the information gain of α in the subset could increase substantially compared to the original dataset. Thus, α would help to distinguish the examples in the subset. Considering multiple patterns, unless the examples are being removed equally in the same portion for every pattern, the information gain for some patterns ought to increase. In this sense, we say the information of a pattern α is *data context sensitive*, and it is incorrect to conclude that less frequent patterns have no information gain.

Another merit of our approach is that, as we select the feature and partition the dataset recursively, the number of features decreases, thus the conditional probability of selecting a discriminative feature increases.

2.4 Non-overfitting

The proposed approach does not overfit and this can be shown in the following discussion as well as experiments. First, various generalization bounds on decision tree is only related to the number of training examples, the depth of the tree and the number of examples at the leaf node (the parameter m in our case), and, importantly, is unrelated to the feature vector. In particular, the bound on balanced trees is the smallest, and $\mathrm{M}^b\mathrm{T}$ is a balanced tree. So the key factor to avoid overfitting is the choice of m. It should not be set too small, such as 1 or 2, like traditional decision tree learning. Second, the support of feature is independent from overfitting. Frequent pattern is containment-based feature and its value is either 0 or 1. A high support pattern has most examples with value 1, while a low support pattern has most 0 values, and this is symmetrical for a classifier. On the other hand, low support features are important. Assuming that the probability of the positive class is 1%. In order to find just one single pattern to separate the two classes as much as possible, the support of this pattern ought to be either close to 99% or 1%. This is trivially true according to pigeon hole principle. In other words, the support of mined pattern is unrelated to generalization.

2.5 Optimality under Exhaustive Search

We study the optimality of $\mathrm{M}^b\mathrm{T}$ under exhaustive search conditions. Assuming that we were able to enumerate all features apriori and use $\mathrm{M}^b\mathrm{T}$ as a "feature selection" algorithm, we show below that the set of features chosen by $\mathrm{M}^b\mathrm{T}$ is still the best set of features.

Under the ideal situation of "exhaustive enumeration", one would consider $\mathrm{M}^b\mathrm{T}$ as a feature selection algorithm since all features are given apriori. Then, the benchmark for comparison would be "feature selection algorithm." Importantly, one would be interested in comparing the quality of "selected features" by $\mathrm{M}^b\mathrm{T}$ with those by bench mark feature selection algorithms. Among the large family of feature selection algorithms, the one that is comparable, is a forward-based feature selection with decision tree or fDT. Assume that the large set of candidate features is $\{f_1, f_2, \ldots, f_N\}$, both the forward-based feature selection fDT and the proposed algorithm $\mathrm{M}^b\mathrm{T}$ selects some K features out of N candidates. To be comparable, the number K is determined by $\mathrm{M}^b\mathrm{T}$ when it reaches its stopping condition, i.e., (1) a node is pure or all the examples belong to the same class (2) the number of examples is $\leq m$.

For forward-based feature selection, assume that there are k features chosen thus far. Then, at the next iteration, it chooses one out of the remaining $N - k$ features to include with the k features that gives the highest accuracy. For simplicity, we assume that the accuracy never decreases before it reaches the parameter K. On the other hand, considering $\mathrm{M}^b\mathrm{T}$, at each iteration, $\mathrm{M}^b\mathrm{T}$ chooses one feature not tested along a decision path from the root to the current node that gives the maximal accuracy increase for that particular subspace of examples within the "current node." Assume that at the end, both M^bT and fDT have each chosen K features independently, as follows, we prove that they choose exactly the same set of K features.

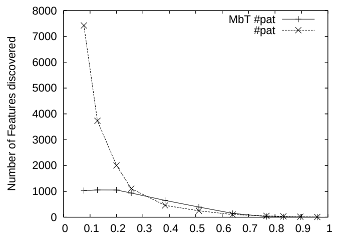
At the first step, obviously both algorithms will choose the same feature and build the same single node tree. Assume after some steps, fDT and M^bT have constructed exactly the same partial tree. We prove that the next feature chosen by both algorithms will be the same and the resulting trees will be identical. First, neither M^bT nor fDT will reconstruct the current partial tree, but rather expand one leaf node. For M^bT, this is true by definition. For fDT, if a new feature would re-construct this partial tree, it would have been chosen previously and already been the testing feature of a non-terminal node of the partial tree. If one would impose that both fDT and M^bT follow the same order on which leaf node to expand next, they would obviously choose exactly the same feature and construct the same tree every step along the way. Nonetheless, this order is not important if fDT satisfies the stopping conditions of M^bT. In fact, when identical nodes from fDT and M^bT get expanded, there is only one unique best feature to choose for both M^bT and fDT. Additionally, a node from both trees does not expand if it satisfies one of the two stopping conditions of M^bT.

3. EXPERIMENTAL STUDIES

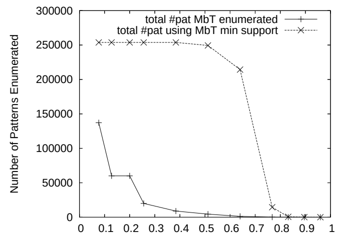
The performance of the proposed algorithm is evaluated on both frequent itemsets and graph datasets. We experimented on some of the most difficult benchmark datasets used in the community and specifically excluded those easier cases. We compared the model-based search tree with closed pattern mining methods (FPClose for itemset [12] and CloseGraph for subgraph [25]) followed by feature selection, as well as a state-of-art integrated "two-step" approach PatClass [5] for frequent itemsets mining. At the time of final copy preparation, a heuristic-based method to directly mine frequent itemsets DDPMine [6] is just published. A comparison on performance is discussed at the end of Section 4. For each dataset, the results reported below are the average of 5-fold CV.

3.1 Itemset Mining

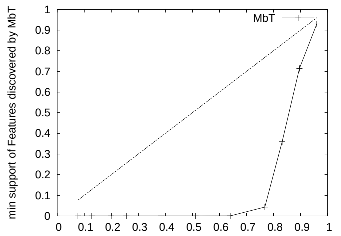
The main concerns on feature discovery algorithms are efficiency and accuracy. For efficiency, it is necessary to find out if the model-based search tree is able to discover the useful needles in the haystack in reasonable amount of time and with reasonable amount of memory. Importantly, one ought to know if traditional methods may even be able to find these predictive patterns given reasonable amount of time and memory. Nonetheless, it is useful to find out the number of additional features that traditional algorithms have to produce before even reaching these patterns. These numbers are important measures of "search quality," i.e., blind-search vs. targeted search. Moreover, the size of the returned feature sets is a substantial quality measure. Obviously, in terms of model comprehensibility, a practitioner would prefer a small number features. Besides various scalability issues, one crucial measure for data mining is the "predictive quality" of the features returned by the model-based search tree. Ideally, one would like to expect an inductive model constructed by those features more accurate or at least as accurate as state-of-the art methods. To answer the above questions regarding the proposed method, we used some of



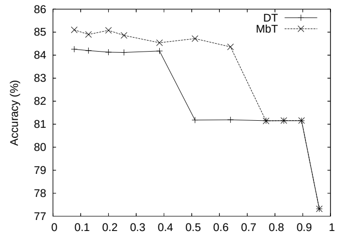
(a) Number of itemsets mined with varying supports



(b) Total number of itemsets enumerated



(c) Minimal supports of itemsets mined by M^bT



(d) Accuracy at different supports

Figure 4: Experiments on Adult; x-axis: Normalized Percentage Support

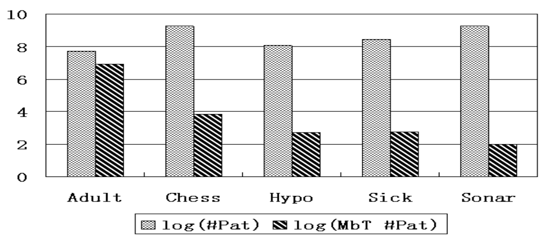
the most difficult benchmark datasets, which are skewed in prior class distribution, and large in scaled problem size in terms of number of examples and feature space. To avoid duplicate and useless patterns, we have employed closed pattern algorithms for both frequent itemset [12] and frequent subgraph mining [25].

Scalability to Mine Frequent Itemsets In Table 1, we summarized the number of frequent itemsets, support employed to mine these itemsets, and most importantly, the minimal support among all those frequent itemsets selected by the proposed method. For model-based search tree, this number is not the same as, but significantly smaller than the minimal support used to invoke the algorithm. For notational convenience, the proposed algorithm is denoted by M^bT, and #pat is the number of patterns or itemsets in this case. Each result column is numbered for convenience.

Among all results, the most interesting numbers to compare and observe are: (1) Column 4 vs. Column 5 - the number of patterns returned by calling frequent pattern algorithm, as compared to calling the proposed algorithm with the same support. (2) Column 3 vs. Column 6 - the minimal support used to generate M^bT as compared to the minimal support of all patterns selected and returned by M^bT.

For the first point, obviously the number of patterns selected by frequent pattern algorithm is much larger (up to $\times 10^3$ larger) than the proposed method. In the left bar chart of Table 1, their normalized "log" scale difference is plotted. We cannot plot it in the original scale since up to 10^3 difference does not demonstrate the result of M^bT. Practically, this difference ought to be interpreted as the evidence that the proposed method is much more selective in choosing the patterns to expand instead of "blind" enumeration. This is obviously due to step 5 of Algorithm 1, that employs a fitness function to maintain the best pattern α at the node, thus split the data into subspaces according to the testing

	1	2	3	4	5	6	7	8	9
DataSet	TrainSize	sup%	sup	#pat	M^bT #pat	M^bT sup	M^bT sup%	#pat using M^bT sup	ratio (5/8)
Adult	39074	20%	7814.8	2245.4	1039.2	10	0.026%	252809	0.41%
Chess	2557	75%	1918	10430.2	46.8	34	1.33%	$+\infty$	$\simeq$ 0%
Hypo	2351	80%	1881	3212	14.8	339.8	7.5%	423439	0.0035%
Sick	2240	85%	1904	4676	15.4	73.6	3.3%	4818391	0.00032%
Sonar	167	25%	42	10623.2	7.4	20	12%	95507	0.00775%



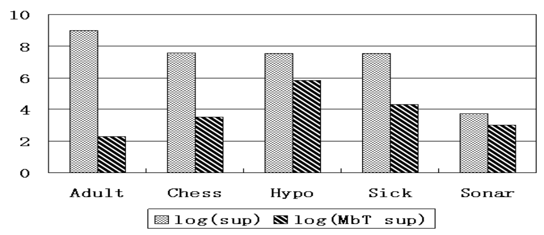


Table 1: Scalability on number of mined itemsets

result on α. This is theoretically analyzed in Section 2.1, and further emphasized by the "predictive quality" of these features discussed thereafter in Section 3.1.

For the second point above, evidently, with a much higher input support (such as 75% for Chess), M^bT can find features whose support are up to $\times 10^2$ lower in value than this invocation parameter (that is 3.3% for Chess). Similarly, their normalized "log" scale difference is plotted in the right bar chart of Table 1. Practically, this means that M^bT, in effect, can "prune" the search space significantly and avoid enumeration of less promising patterns.

Solving Combinatorial Explosion In Table 1, the number of patterns (column 8) returned by closed frequent pattern mining method using the minimum support among all features selected by M^bT is summarized for each dataset. The ratio of this number as compared to the number of features selected by M^bT (column 5) is calculated in column 9. Unequivocally, their difference is at least from $\times 10^3$ to $\times 10^6$. Importantly, on Chess, it is impossible for the closed pattern mining algorithm to finish using the minimal support returned by M^bT. Due to combinatorial explosion, the program simply "ate" all the memory that the Linux server could allocate, and was then killed by the operating system.

In order to clearly demonstrate the scalability of the proposed method in avoiding enumerating huge number of potentially useless features, but drilling down quickly to patterns with extremely low support, we ran several additional experiments on Adult with varying supports. In Figure 4(a), we compared the number of patterns mined by closed frequent itemset algorithm and M^bT as the support goes down from 0.95 and 0.05. Clearly, the number of patterns returned by M^bT exhibits linear-like growth as a function of $1 - sup$. However, for traditional closed pattern mining algorithm, the plot appears to be exponential in shape.

As a different way to demonstrate scalability, Figure 4(b) plots the total number of patterns ever "enumerated" by M^bT (including all patterns enumerated at each non-leaf node), as well as the total patterns generated by state-of-the-art closed pattern algorithm using the minimal support of all features selected by M^bT. The latter number could have been much larger if we had used the minimal support of all features "enumerated" by M^bT. It is evident that the growth on the total number of patterns ever enumerated

DataSet	DT #pat	M^bT #pat	DT Acc	M^bT Acc
Adult	2245.4	1039.2	84.14%	**85.08%**
Chess	10430.2	40.8	72.85%	**80.30%**
Hypo	3212.2	14.8	**99.02%**	97.9%
Sick	4676	15.4	96.68%	**97.38%**
Sonar	10623.2	7.4	79.80%	**80.78%**

Table 2: Accuracy on mined itemsets vs. bigger sets

	with original features		w/o original features		feature set size	
Data Set	M^bT	Pat-Class	M^bT	Pat-Class	M^bT	Pat-Class
adult	0.853	0.848	0.853	0.761	1039.2	45.6
chess	0.884	0.871	0.82	0.628	25.4	13.2
hypo	0.993	0.993	0.979	0.992	24.2	15.8
sick	0.973	0.974	0.975	0.965	16.6	29.2
sonar	0.803	0.818	0.764	0.818	7.4	24.4

Table 3: Accuracy of SVM on mined itemsets

by M^bT is much smaller than that of closed pattern mining method with the same support.

Yet, the third way to demonstrate the scalability is to examine the variation of the minimal support among all selected featured by M^bT as the invocation support changes. As plotted in Figure 4(c), when the input support decreases, the minimal support returned by the proposed algorithm quickly goes down to nearly zero (after the invocation support is less than 60%). This explains "the quick convergence to high accuracy" as discussed below.

Convergence Speed One practical concern is that one does not wish to experiment very low support before the accuracy converges, but rather prefer a method that converges fast and at high support. In Figure 4(d), we plotted the changes in accuracy as the input support goes down, comparing both the proposed method M^bT and a decision tree trained from the larger pattern sets returned by traditional closed pattern mining method. It is quite straightforward to see that at high support 70%, the proposed method already reaches promising accuracy, which can only be achieved by traditional approach at support $< 10\%$.

	1	2	3	4	5	6	7	8	9	10	11
		Positive				DT	M^bT	M^bT	M^bT	#pat using	ratio
DataSet	#Training	Dist	#Test	sup	sup%	#pat	#pat	sup	sup%	M^bT sup	(7/10)
NCI1	33020	1.0%	8254	9906	30%	921	77	56	0.17%	$+\infty$	$\simeq 0\%$
NCI33	31991	4.1%	7997	4799	15%	630	344	32	0.10%	$+\infty$	$\simeq 0\%$
NCI41	22008	5.7%	5501	2201	10%	1579	376	15	0.068%	$+\infty$	$\simeq 0\%$
NCI47	32217	5.0%	8054	3222	10%	1609	587	10	0.031%	$+\infty$	$\simeq 0\%$
NCI81	32426	5.9%	8106	3243	10%	1594	685	10	0.031%	$+\infty$	$\simeq 0\%$
NCI83	22217	8.3%	5553	2222	10%	1583	620	10	0.045%	$+\infty$	$\simeq 0\%$
NCI109	32424	5.1%	8102	3242	10%	1602	605	10	0.031%	$+\infty$	$\simeq 0\%$
NCI123	31812	7.9%	7953	3181	10%	1616	909	10	0.031%	$+\infty$	$\simeq 0\%$
NCI145	32004	4.9%	8000	3200	10%	1591	491	15	0.047%	$+\infty$	$\simeq 0\%$
H1	34118	3.5%	8528	3412	10%	1265	504	19	0.056%	$+\infty$	$\simeq 0\%$
H2	33256	1.0%	8312	3326	10%	1248	156	10	0.030%	$+\infty$	$\simeq 0\%$

Table 4: Number of mined subgraphs on Graph Data Set

Accuracy of Mined Itemsets Predictive quality of the mined compact feature sets is measured against both (1) much larger feature sets, and (2) those mined feature sets by state-of-the-art "two-step" approaches. For the first, the accuracy of M^bT's feature is compared against the much larger feature sets returned by closed pattern algorithm, both with the same invocation support. Clearly, as summarized in Table 2, except for hypo, the much smaller feature set ($\times 10^2$ to $\times 10^3$ smaller) can actually produce more accurate model than a much larger feature set and the accuracy increase is up to 8%. This clearly demonstrate both the predictive and comprehensible quality of mined features.

To further justify the predictive quality, we have also compared M^bT with the most recently proposed closed pattern mining algorithm, PatClass [5]. One important understanding is that other feature discovery algorithm can be called as the baseline pattern searching algorithm by M^bT. In a way, none of them are competing algorithm for the proposed approach, but they can be "plugged" in together to find even better patterns than the closed frequent itemset algorithm solely used in the experimental study. Nonetheless, the motivation of this comparison with PatClass is to demonstrate that even M^bT calls the "non-feature selective" (i.e, no feature selection and no pruning of search space) closed pattern mining algorithm, it can still reach or exceed the performance of the best "selective" two-step approach that we are aware of. The results using SVM as the inductive learner, with or without the original feature vector, are summarized in Table 3. As can be clearly shown, their accuracy are quite close. Both feature sets are small, and normally, a bigger feature set incurs higher accuracy due to more expressive power.

3.2 Graph Mining

Frequent-subgraph based graph mining has been the recent active topic to use frequent pattern concept to mine predictive subgraphs as features, thus producing accurate inductive models that can be used in drug design, social network analysis, etc. In normal understanding, frequent-subgraph mining is more difficult than frequent itemsets since the scaled problem size is usually much larger and the graph isomorphism test itself is a non-trivial research problem. Thus, frequent-subgraph mining provides an exciting test bed for the proposed method.

From PubChem project [2], we selected a series of graph datasets with rather skewed distributions. As commonly recognized by the graph mining community, these are some of the most challenging tasks. Each of the NCI anti-cancer screens forms a classification problem, where the class labels are either active or inactive. The active class is very rare compared with the inactive class. Another dataset is obtained from the AIDS anti-viral screen program [1]. The screening tests are evaluated in one of the following three categories: confirmed active (CA), confirmed moderately active (CM) and confirmed inactive (CI). Both CA and CM classes are extremely rare compared with CI. Two classification problems are formulated out of this dataset. The first problem is designed to classify between CM+CA and CI, denoted as H1; the second between CA and CI, denoted as H2. The characteristic of each graph dataset is described in columns 1-3 of Table 4.

Scalability to Mine Frequent Subgraphs Table 4 summarizes the number of subgraphs mined by traditional closed graph mining algorithm (column 6) and the proposed method (column 7). Over all graph sets, the number of subgraphs selected by closed graph mining algorithm is at least two times and up to eleven times greater than that of M^bT. Another most remarkable pair of columns are columns 5 and 9, which are respectively the input support and the minimal support among all subgraphs selected by M^bT. Their magnitude of difference is from $\times 10^2$ to $\times 10^3$.

Solving Combinatorial Explosion for Frequent Subgraph Mining We also examined if the closed graph mining algorithm is able to generate any subgraphs if the input support is chosen to be the minimal support of all subgraphs selected by M^bT. However, as indicated in column 10 of Table 4, the program consumed all memory that could be allocated by Linux, and none of them could finish. This not only justifies the scalability of the proposed algorithm on frequent subgraph mining, but also provides a solution to combinatorial explosion for the same context.

Accuracy of Mined Frequent Subgraphs Similar to frequent itemsets, performances of M^bT are compared with a classifier (E.g. DT) trained with much larger number of frequent subgraphs returned by the closed subgraph mining algorithm with the same invocation support. The results of the same classifier built on the mined subgraphs by M^bT are also reported. Using the subgraphs selected by M^bT (columns 7 in Table 4), and the much larger set of subgraphs mined by the closed graph mining algorithm (columns 6 in Table 4), Table 5 summarizes the AUC and accuracy results of M^bT and a decision tree. Since these datasets are rather

Data Set	DT AUC	M^bT AUC	DT M^bT AUC	DT Acc	M^bT Acc	DT M^bT Acc
NCI1	0.61	**0.685**	**0.74**	0.989	**0.99**	0.989
NCI33	0.726	**0.743**	**0.745**	0.946	**0.948**	**0.95**
NCI41	0.738	**0.765**	**0.763**	0.937	**0.942**	**0.938**
NCI47	0.718	0.708	**0.727**	0.938	**0.943**	**0.939**
NCI81	0.693	**0.696**	**0.723**	0.923	**0.933**	**0.93**
NCI83	0.712	**0.734**	**0.722**	**0.910**	0.890	0.890
NCI109	0.744	0.699	**0.746**	0.935	0.934	**0.938**
NCI123	0.678	0.667	**0.679**	0.894	0.892	**0.91**
NCI145	0.749	0.747	**0.752**	0.938	**0.953**	**0.948**
H1	0.637	**0.675**	**0.667**	0.949	**0.965**	**0.965**
H2	0.681	**0.707**	**0.695**	0.988	0.988	**0.989**

Table 5: Performances of DT,M^bT and DT M^bT

	Proposed Method ($\geq 10\%$)		SVM Bchmk 5% + fs		C4.5 Bchmk 5% + fs	
Data Set	M^bT	DT M^bT	org	rBlcd	org	rBlcd
NCI1	0.685	**0.74**	0.583	0.736	0.589	0.65
NCI33	**0.743**	**0.745**	0.512	0.737	0.536	0.648
NCI41	**0.765**	**0.763**	0.679	0.72	0.603	0.606
NCI47	0.708	0.727	0.501	**0.75**	0.63	0.64
NCI81	0.696	0.723	0.541	**0.739**	0.589	0.652
NCI83	**0.734**	**0.722**	0.633	0.692	0.594	0.608
NCI109	0.699	**0.746**	0.508	0.727	0.555	0.64
NCI123	**0.667**	**0.679**	0.517	0.619	0.606	0.608
NCI145	0.747	0.752	0.55	**0.755**	0.595	0.654
H1	**0.675**	**0.667**	0.632	0.661	0.399	0.556
H2	0.707	0.695	0.519	**0.845**	0.427	0.682

Table 6: AUC of M^bT, DT M^bT vs. Benchmarks

skewed, AUC or area under curve is a more appropriate measure than accuracy only. As highlighted in the table, the proposed method (M^bT) and the decision tree built on the mined subgraphs by M^bT (DT M^bT) have achieved higher AUC and accuracy in almost all of the graph data than the decision tree constructed on the larger subgraph sets (DT). Across all datasets, the average improvement in AUC is 0.04 or 4%. Importantly, the most significant improvement, 21%, is achieved on the most skewed dataset NCI1 (1% positive).

We have also compared the AUC of the proposed method with two benchmark results where the graphs are generated with the batch two step approach: first enumerating closed graphs with support 5% and then use feature selection to choose the top 1000. The results are summarized in Table 6. There are two benchmark methods involved. Among them, "org" is trained on the frequent subgraphs mined from the original skewed training set. On the other hand, "rBlcd" use the subgraphs mined from a "rebalanced" sample where skewed positives are always kept, but negatives are down-sampled. Obviously, over all datasets, the AUC scores achieved by M^bT and the decision tree built on the mined subgraphs by M^bT (DT M^bT) consistently dominate those of org and rBlcd via C4.5. For seven out of eleven graph sets, M^bT or DT M^bT performs significantly better than org and rBlcd based on SVM.

4. RELATED WORK

The usage of frequent pattern in classification has been explored by many recent studies. The association between frequent patterns and class labels is used for prediction. Earlier studies on associative classification [19, 18, 27] mainly focus on mining high-support, high-confidence rules and building a rule-based classifier. Prediction is made based on the top ranked rule or multiple rules. A recent work on top-k rule mining [7] discovers top-k covering rule groups for high-dimensional gene expression profiles. A classifier RCBT is constructed from the top-k covering rule groups and achieves very high accuracy. Harmony [23] is another rule-based classifier which directly mines classification rules. It uses an instance-centric rule-generation approach and assures for each training instance, that one of the highest-confidence rules covering the instance is included in the rule set. In addition, [5] is a newly proposed frequent pattern-based classification method. Highly discriminative frequent itemsets are selected to represent the data in a feature space, based on which learning algorithm can be used for model learning. With no initial feature vector representation, the primary problem in classification of complex data such as graphs is feature invention. In recent years, much work has been carried out to address the graph classification problem. Basically these studies can be divided into three approaches: (1) structure or fragment-based approach [15, 9, 22], (2) kernel-based approach [20, 10], and (3) boosting method [16]. Typically, the basic idea of structure or fragment-based approach is to extract frequent substructures [15, 9], local graph fragments [22], or cyclic patterns and trees [14] and use them as descriptors to represent the graph data. Studies with kernel-based approach aim at designing effective kernel functions to measure the similarity between graphs.

Several recent proposals have discussed how to make frequent pattern mining more conscious of memory hierarchy and architecture, including [11, 8]. [11] proposed a cache-conscious prefix tree which improves spacial locality and enhances the benefits from hardware cache line prefetching. [8] proposed a parallel mining algorithm of sequential patterns on a distributed memory system. These techniques are related to our mining task and can be applied to further improve the efficiency of the proposed method. Besides these efficient algorithms on the architecture level, there are some up-to-date methods DDPMine[6] and LEAP[24] on algorithm level which directly mines the most discriminative pattern via specially designed heuristics without mining the complete set of frequent patterns. Since a lot of search space can be pruned, these methods can still find many of the most discriminative patterns, but are much more efficient than traditional frequent pattern mining methods. The difference of this paper from DDPMine and LEAP is that the proposed techniques are applicable to frequent patterns in general, not limited to only either itemsets (DDPMine) or sub-graphs (LEAP) [24]. The accuracy of DDPMine mined itemsets as reported in Table IV of [6] are comparable and very similar to those numbers in Table 3 of this paper on M^bT. Similar to the closed pattern mining algorithms called by M^bT in this paper, M^bT can also invoke the most recently proposed methods DDPMine (or LEAP) at its internal node to mine candidate features to split its data space.

5. CONCLUSION

To solve the scalability issue of mining frequent pattern as feature vectors from semi-structured and unstructured data, traditional methods employ a two-step batch process that first enumerates all candidate features, then performs feature selection. This process has limited success in identifying a small and compact set of features. Furthermore, different techniques are re-invented to reduce the search space for different types of patterns: frequent itemsets, frequent subgraphs, and sequential patterns. In other words, each technique is hard to generalize across different problems. To address these problems and others discussed in the paper, we propose a divide-and-conquer approach. The proposed method constructs a model-based search tree as it recursively invokes some frequent pattern enumeration algorithm. The main idea is to mine a discriminative feature that divides a subset of examples into purer subspaces that previously chosen patterns fail to distinguish. This process recursively runs on smaller and smaller data subspaces until either the subspace is too small or every example in the subspace belongs to the same class. At the end of feature discovery process, we have both a predictive decision tree and a set of discriminative features kept in non-leaf nodes of the tree. The proposed method can mine predictive patterns with extremely low global support, scales linearly to the scaled problem size and does not overfit.

Experimental studies have been conducted on both frequent itemset and graph mining problems. We have selected some of the most difficult datasets in each field. For example, some graph datasets have highly skewed distribution (1% positives) and the chemical compounds have hundreds of edges and vertices. The baseline algorithms invoked by model-based search tree are basic closed pattern algorithms. For scalability, the total number of patterns both enumerated during the pattern mining process and finally selected by the proposed algorithm is up to 10^3 smaller than those generated by the baseline algorithm using the same input support. The minimal support of the patterns discovered by the proposed algorithm can be so low (such as, 0.03%) that calling the closed pattern mining algorithms to enumerate these features is impossible due to combinatorial explosion and resource constraints. For predictive quality of those mined patterns, on the frequent itemset data, the accuracy is as good as most recently proposed methods, and significantly better than a model constructed from the large set of patterns discovered by traditional closed pattern mining. On the challenging skewed graph mining problem, the AUC using the mined subgraphs is up to 21% higher than comparable benchmark methods.

Future Work: (1) We generated biased dataset and found that M^bT can still find good features even when the training and testing data follow significantly different prior class distribution. More studies are being conducted. (2) M^bT is a scalable frequent pattern mining algorithm not limited to just itemsets and sub-graphs. It is interesting to systematically compare M^bT with heuristic-based scalable algorithm designed for different types of frequent patterns. To compare with DDPMine [6] on itemsets and with LEAP [24] on graphs, M^bT can either invoke closed pattern methods [12, 25] or call DDPMine/LEAP instead at internal nodes.

Acknowledgment

The research of Kun Zhang was partially funded by Louisiana Cancer Research Consortium.

6. REFERENCES

[1] http://dtp.nci.nih.gov. The Aids Antiviral Screen.

[2] http://pubchem.ncbi.nlm.nih.gov. The PubChem Project.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB*, pages 487–499, 1994.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of ICDE*, pages 3–14, 1995.

[5] H. Cheng, X. Yan, J. Han, and C. Hsu. Discriminative frequent pattern analysis for effective classification. In *Proc. of ICDE*, 2007.

[6] H. Cheng, X. Yan, J. Han, and P. Yu. Direct discriminative pattern mining for effective classification. In *Proc. of ICDE*, 2008.

[7] G. Cong, K. Tan, A. Tung, and X. Xu. Mining top-k covering rule groups for gene expression data. In *Proc. of SIGMOD*, pages 670–681, 2005.

[8] S. Cong, J. Han, and D. Padua. Parallel mining of closed sequential patterns. In *KDD*, 2005.

[9] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. Knowl. and Data Eng.*, 17(8):1036–1050, 2005.

[10] H. Fröhlich, J. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *Proc. of ICML*, pages 225–232, 2005.

[11] A. Ghoting, G. Buehrer, S. Parthasarathy, D. Kim, A. Nguyen, Y. Chen, and P. Dubey. Cache-conscious frequent pattern mining on a modern processor. In *VLDB*, 2005.

[12] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, 2003.

[13] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. of SIGMOD*, pages 1–12, 2000.

[14] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proc. of KDD*, pages 158–167, 2004.

[15] S. Kramer, L. Raedt, and C. Helma. Molecular feature mining in hiv data. In *Proc. of KDD*, pages 136–143, 2001.

[16] T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Proc. of NIPS*, pages 729–736, 2004.

[17] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. of ICDM*, pages 313–320, 2001.

[18] W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *Proc. of ICDM*, pages 369–376, 2001.

[19] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. of KDD*, 1998.

[20] P. Mahë, N. Ueda, T. Akutsu, J. Perret, and J. Vert. Extensions of marginalized graph kernels. In *Proc. of ICML*, pages 552–559, 2004.

[21] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. of ICDE*, pages 215–226, 2001.

[22] N. Wale and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proc. of ICDM*, pages 678–689, 2006.

[23] J. Wang and G. Karypis. HARMONY: Efficiently mining the best rules for classification. In *Proc. of SDM*, pages 205–216, 2005.

[24] X. Yan, H. Cheng, J. Han, and P. Yu. Mining significant graph patterns by leap search. In *SIGMOD*, 2008.

[25] X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *KDD*, 2003.

[26] G. Yang. Computational aspects of mining maximal frequent pattern. *Theoretical Computer Science*, 2006.

[27] X. Yin and J. Han. Cpar: Classification based on predictive association rules. In *Proc. of SDM*, 2003.