Exploring Constraints to Efficiently Mine Emerging Patterns from Large High-dimensional Datasets

Xiuzhen Zhang
The University of Melbourne
xzhang@cs.mu.oz.au

Guozhu Dong
Wright State University
gdong@cs.wright.edu

Ramamohanarao Kotagiri
The University of Melbourne
rao@cs.mu.oz.au

ABSTRACT

Emerging patterns (EPs) were proposed recently to capture changes or differences between datasets: an EP is a multivariate feature whose support increases sharply from a background dataset to a target dataset, and the support ratio is called its growth rate. Interesting long EPs often have low support; mining such EPs from high-dimensional datasets is a great challenge due to the combinatorial explosion of the number of candidates. We propose a Constraint-based EP Miner, ConsEPMiner, that utilizes two types of constraints for effectively pruning the search space: External constraints are user-given minimums on support, growth rate, and growth-rate improvement to confine the resulting EP set. Inherent constraints — same subset support, top growth rate, and same origin — are derived from the properties of EPs and datasets, and are solely for pruning the search space and saving computation. ConsEPMiner can efficiently mine *all* EPs at low support on large high-dimensional datasets, with low minimums on growth rate and growth-rate improvement. In comparison, the widely known Apriori-like approach is ineffective on high-dimensional data. While ConsEPMiner adopts several ideas from Dense-Miner [4], a recent constraint-based association rule miner, its main new contributions are the introduction of inherent constraints and the ways to use them together with external-constraints for efficient EP mining from dense datasets. Experiments on dense data show that, at low support, ConsEPMiner outperforms the Apriori-like approach by orders of magnitude and is more than twice as fast as the Dense-Miner approach.

1. INTRODUCTION

Emerging patterns (EPs) are a new type of knowledge recently introduced [5] to capture emerging trends in time-stamped databases, or useful contrasts between data classes. Given a background dataset D' and a target dataset D'', the *growth rate* of an itemset X from D' to D'' is defined as $GR(X) = \frac{supp''(X)}{supp'(X)}$ (we define $\frac{0}{0} = 0$ and $\frac{>0}{0} = \infty$), where $supp'(X)$ and $supp''(X)$ denote the support[1] of X in D' and D'' respectively. Given a growth rate threshold $\rho > 1$, an *emerging pattern* from D' to D'' is an itemset whose growth rate from D' to D'' is $\geq \rho$. When D' is clear from context, an EP X from D' to D'' is simply called an EP of D''. The support of X in D'', $supp''(X)$, is called *the support of the EP*. The support and growth rate of an EP describe its applicability and strength.

Example 1. The following are two EPs from the Malignant class to the Benign class of the Wisconsin-breast-cancer dataset from the UCI machine learning repository.

EP	Malignant-support	Benign-support	growth rate
e_1	0.41%	20.31%	49.54
e_2	0%	3.28%	∞

e_1={(bare-nuclei,1),(bland-chromatin,3),(norm-nucleoli,1),(mitoses,1)}
e_2={(marginal-adhesion,1),(bare-nuclei,1),(normal-nucleoli,2)}

The EP e_1, with a growth rate of 49.54, is a four-attribute feature contrasting benign instances against malignant instances. It has a very high predictive power: The odds that instances containing (or satisfying) e_1 are benign is 98%. The EP e_2 has even greater predictive power: The odds that instances containing e_2 are benign is 100%.

EPs are characteristics distinguishing the target dataset from the background dataset. We have built powerful classification systems [7] by aggregating the differentiating power of EPs and the resulting classifiers are usually more accurate than existing state-of-the-art classifiers. Besides classification, EPs are also useful as predictive patterns.

Differing from market basket type applications, datasets for medicine, government, science and business are very often not only large, but also of high dimensions and *dense* (i.e. each instance has a value for each dimension). In the UCI Connect-4 dataset, the Win and Loss classes consist of 16,635 and 67,557 instances respectively, and each instance is defined by 42 attributes (and thus a transaction consists of 42 items). Such datasets usually "contain" a huge number of EPs, and many of them are long (consisting of tens of items). The Apriori-like [1] approach, considering only the support constraint at the mining stage, are ineffective on such datasets: the support constraint cannot control the combinatorial explosion of candidate itemsets; the bottom-up itemset lattice framework is very expensive for searching

[1] Let U denote the item space. An *itemset* X is a subset of U. A *transaction* is an itemset and a dataset is a collection of transactions. The *support* of an itemset X in a dataset D, $supp(X)$, is $\frac{|\{t \in D | X \subseteq t\}|}{|D|}$.

long EPs — to mine an EP of 30 items, whatever pruning techniques are used, at least $2^{30} \approx 10^9$ candidate itemsets, all subsets of the EP, have to be generated and tested!

This paper proposes an algorithm *Constraint-based EP Miner*, *ConsEPMiner* for short, for efficiently mining EPs satisfying given support and growth-rate thresholds. Among other ideas that will be discussed later, it uses both the support and growth rate constraints to directly prune the search space. We also introduce the *growth-rate improvement* constraint to eliminate EPs that are uninteresting, and use it for direct pruning.

Minimums on support, growth rate and growth-rate improvement are all *external constraints* set by users to restrict the resulting EP set. We also consider additional constraints that can help further prune the search space and save computation; some of these constraints are implied by the datasets, and others are implied by special properties of EPs. We call these constraints the *inherent* constraints, since they are not given by users and do not affect the resulting EP set.

In contrast to the Apriori-like approach, ConsEPMiner effectively controls the explosion of candidates from the very beginning of mining, and achieves high efficiency on large high-dimensional datasets with modest support, growth rate and growth-rate improvement minimums. It achieves this by exploiting both external and inherent constraints.

Our work is motivated by Dense-Miner[4], a recently proposed external-constraint-based association rule mining algorithm that is much more effective than the Apriori-like approach on dense data. However, ConsEPMiner considers new issues specific to EP mining, and more importantly, uses new ideas based on inherent constraints. In ConsEPMiner we use the following ideas from Dense-Miner: the set-enumeration tree (SE-tree) seach framework (Section 3) and the breadth-first search strategy (Section 4), the bounding mechanism for applying user-given minimums for pruning, and the reordering considerations for more pruning chances. Based on the external constraints, we introduce several new ideas on effectively using inherent constraints to help prune the search space. Although the inherent constraints do not further restrict the final resulting EP set, considering them explicitly reduce the overhead of computing group bounds and scanning datasets, and the inherent-constraint-based tail item reordering heuristics produces more pruning chances.

To see the benefit of considering inherent constraints explicitly, we developed EP-DenseMiner. As in Dense-Miner, it uses all the external constraints for direct pruning and adopts largely the same pruning techniques. To mine EPs, it uses the bounding theorems of Section 3.2 and the external-constraint-based item reordering heuristics of Section 3.3. Conceivably, EP-DenseMiner suffers from the overhead of not considering the inherent constraints explicitly and misses the additional pruning chances produced by inherent constraints. Experiments indeed show that ConsEPMiner consistently outperforms EP-DenseMiner.

Related works: In [2], an Apriori-like algorithm for mining association rules was proposed, where some inherent constraints are considered. Our experiments show that the Apriori-like EP miner, even enhanced with all the inherent constraints, is still ineffective on high-dimensional data and is far inferior to ConsEPMiner. [5] proposed algorithms for finding the border description (but without actual support and growth rate) for EPs satisfying given support and growth rate minimums. Further processing is needed to extract the embodied EPs and their support and growth rate, and to select the interesting ones. The border finding algorithms (Max-Miner [3] or HorizonMiner [6]) only use the support constraint in mining; the border manipulation algorithms do not explicitly use constraints. In [7], an idea similar to growth-rate improvement was used to reduce the EP classifier, but it was not exploited to make mining more efficient. In [9] and [8], external constraints of various types, orthogonal to those considered here, are exploited to mine constrained association rules. The idea of pruning in an SE-tree search employed in Dense-Miner and ConsEPMiner originates from the idea of pruning in a lattice space with pruning functions of [11].

2. CONSTRAINTS IN EP MINING

To efficiently mine EPs, in addition to support and growth rate, we also propose *growth-rate improvement* and *inherent* constraints. Minimums on support ($minsupp$), growth rate ($minrate$) and growth-rate improvement ($minrateimp$) are called *external* constraints. They ensure a final set of EPs with sufficient predictive power and support and where none of them is redundant; importantly, they are also used to directly prune the search space. Inherent constraints are introduced only for pruning the search space and saving computation.

2.1 Growth-rate improvement

The growth-rate improvement constraint is concerned with the subset relationship between EPs and aims to remove EPs that are not essential.

Example 2. Consider two EPs of the Benign class of the Wisconsin-breast-cancer dataset.

EP	Benign support	growth rate
e_3={(clump-thickness,1), (unif.-cell-shape,3)}	2.62%	6.32
e_4={(clump-thickness,1)}	31.0%	24.9

With a growth rate of 6.32, e_3 seems to be an interesting EP. However, compared to e_4, e_3 is actually significantly inferior in predictive ability (e_4 has much higher growth rate), conciseness ($e_4 \subset e_3$), and applicability (e_4 has much higher support).

In [4], improvement of association rules is defined in terms of the difference in confidence of rules. We define below the growth-rate improvement of EPs [2].

Definition 1. Given an EP e, the growth-rate improvement of e, $rateimp(e)$ is defined as the minimum difference between its growth rate and the growth rate of all of its subsets,

$$rateimp(e) = min(\forall e' \subset e,\ GR(e) - GR(e'))$$

A positive growth-rate improvement threshold, $minrateimp > 0$, ensures a concise, "grid-like" representative set of EPs which are not subsumed by one another and where each EP

[2]This constraint was also independently used in [7].

consists of items that are strong contributors to its predictive power. Experiments show that modest *minrateimp* settings can reduce the resulting EP set dramatically: For the UCI Mushroom dataset, at $minsupp = 20\%$, $minrate = 1.01$, a modest setting of $minrateimp = 0.01$ reduces the number of EPs of the Edible class from 14,342 to 354.

Growth-rate Improvement is in terms of the absolute difference on growth rate. We now define a similar constraint in terms of the relative difference on growth rate.

Definition 2. Given an EP e, the relative growth rate improvement of e, $rel\_rateimp(e)$ is defined as the minimum ratio of its growth rate over that of all its subsets,
$rel\_rateimp(e) = min(\forall e' \subset e \ s.t. \ GR(e') > 0, \ GR(e)/GR(e'))$

Given an EP e, $rel\_rateimp(e) > 1$ implies $rateimp(e) > 0$. The advantage of relative growth-rate improvement is that it allows us to prune uninteresting EPs based on their strength relative to that of their subsets. We will only report experimental results using *minrateimp*; we note that ConsEP-Miner is faster in experiments using the *rel_rateimp* threshold and produces smaller EP sets than using *minrateimp*.

2.2 Inherent constraints

Inherent constraints, namely *same subset support*, *top growth rate* and *same origin*, are natural restrictions formed by the properties of EPs or datasets on which itemsets can be valid EPs. As will be discussed in the next section, these constraints provide new perspectives on pruning the search space and saving computation.

Same subset support: We say that an itemset S satisfies the *same subset support* constraint if there exists $X \subset S$ such that S has the same support as X, and we denote the fact by $sameSupp(X, S)$. Note that if S satisfies the constraint due to X, then so does $S \cup Y$ (due to $X \cup Y$). The constraint has two forms: (1) *same target support* — $supp''(X) = supp''(S)$, denoted as $sameSupp''(X, S)$; (2) *same subset background support* — $supp'(X) = supp'(S)$, denoted as $sameSupp'(X, S)$. We use the constraint in two ways: (1) Given $sameSupp''(X, S)$, suppose $supp''(X)$ is known, we can derive $supp''(S)$ and reduce dataset scanning time. (2) Given $sameSupp'(X, S)$, since $supp''(X) \geq supp''(S)$, and $GR(X) \geq GR(S)$, as a result, $rateimp(S) \leq 0$ and thus S and its supersets can be pruned.

Top growth rate: We denote by $topGR(X)$ the fact that X has growth rate ∞ (i.e., X is a jumping EP [6]). Obviously, all supersets of a jumping EP have growth-rate improvement ≤ 0, and thus can be pruned. The top growth rate constraint is more useful at low support, since more jumping EPs appear there.

Same origin: In the binary dataset mapped from relational data, items are mapped from (`attribute, interval`) or (`attribute, value`) pairs. Itemsets containing items mapped from the same attribute, and their supersets, cannot be EPs and should be pruned. We call this constraint the *same origin* constraint and use $sameOrg(X, i)$ to denote that the constraint holds between some item of X and item i.

As will be seen in Section 3, the monotone property of inherent constraints allows us to collect the constraints and prune groups of hopeless itemsets at the early stages of mining.

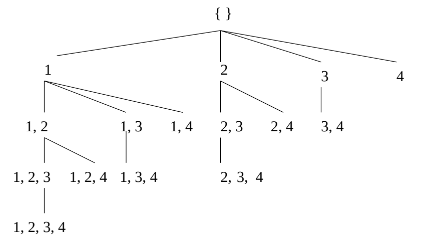

Figure 1: A complete set enumeration tree over U={1, 2, 3, 4}, with items lexically ordered.

3. PRUNING WITH CONSTRAINTS

We consider how to best utilize external and inherent constraints to reduce the search space and save time for database scanning and CPU processing. We will use the notations developed in Dense-Miner [4] in later discussions.

Our underlying itemset search framework is the set enumeration tree (SE-tree) [10]. As shown in Figure 1, by imposing an ordering on items, all itemsets in the item space are enumerated. A node g of an SE-tree is represented as a *group* comprising two itemsets: *head*, $h(g)$, the itemset enumerated at g, and *tail*, $t(g)$, an ordered set consisting of items that can potentially be appended to $h(g)$ to form an itemset enumerated by some sub-node of g. For example in Figure 1, for the root node r, $h(r) = \phi$, $t(r) = \{1, 2, 3, 4\}$. A child g_c of g is formed by taking an item $i \in t(g)$ and appending it to $h(g)$ to get $h(g_c)$; $t(g_c)$ then contains all items in $t(g)$ that follow i in the ordering. Given this child expansion policy, without any pruning of nodes or items, the SE-tree enumerates all possible itemsets. The order in which items appear in $t(g)$ is significant since it reflects how its children are to be expanded. An itemset e is *derivable* from a group g if $h(g) \subset e$, and $e \subseteq h(g) \cup t(g)$. To process a group g in a dataset D, we scan D to compute the support of $h(g)$, $h(g) \cup \{i\}$ ($\forall i \in t(g)$), and $h(g) \cup t(g)$. In EP mining, given a background dataset D' and a target dataset D'', g is *half processed* if g is processed in D''; g is *completely processed* if it is also processed in D'. We scan the SE-tree breadth-first for valid EPs. In this process, as proposed in [4], while pruning entire groups, we also prune group tail items.

3.1 Pruning first with inherent constraints

We adopt the "pruning first with inherent constraints" strategy to prune unpromising groups of an SE-tree, specifically, we directly prune group tail items with inherent constraints.

Observation 1. *Given a group g and an item $i \in t(g)$, i can be pruned if (1) $sameSupp'(h(g), h(g) \cup \{i\})$, or (2) $topGR(h(g) \cup \{i\})$, or (3) $sameOrg(h(g), i)$.*

The observation follows directly from the inherent constraint definitions. More importantly, with the breadth-first search of SE-tree, checking the satisfiability of inherent constraints is a very minor computation and does not incur extra scanning over datasets. Only after pruning the tail items of a group with inherent constraints, will we prune the group with the more costly external constraints. This strategy reduces the number of groups that must be considered for pruning with external constraints, and the number of groups whose frequency must be counted during database scans.

In addition to pruning, inherent constraints are also useful for reducing database scanning time.

OBSERVATION 2. *Given a group g and $i \in t(g)$, in generating its child group g_c where $h(g_c) = h(g) \cup \{i\}$ and $t(g_c) = \{j \in t(g) | j$ follows $i\}$, (1) if $sameSupp''(h(g), h(g) \cup \{i\})$, then $supp''(h(g_c) \cup \{j\}) = supp''(h(g) \cup \{j\})$; (2) if $sameSupp''(h(g), h(g) \cup \{j\})$, then $supp''(h(g_c) \cup \{j\}) = supp''(h(g) \cup \{i\})$.*

During the breadth-first search of an SE-tree, by applying this observation in generating new groups, we can derive the target support of tail items of many child groups of the next level directly and avoids the expensive database scanning to get their support.

3.2 Pruning with external constraints

In Dense-Miner, for mining association rules, to prune groups with external constraints, theorems were proposed to upper bound the support, confidence and confidence improvement of groups. In EP mining, as the support, growth rate and growth-rate improvement of EPs are defined differently, we need to develop similar but new theorems bounding groups. Specifically, for a group g, we compute the upper bounds of support ($usupp(g)$), growth rate ($urate(g)$) and growth-rate improvement ($urateimp(g)$).

$usupp(g)$ is relatively easy to compute because support is anti-monotone. ConsEPMiner estimates $usupp(g)$ by $min(supp''(X)|X \subseteq h(g))$. The estimation of other bounds requires more costly computation.

THEOREM 1. *Given a group g, $urate(g) = \frac{x''_m}{x'_n}$, where $x''_m \geq supp''(h(g))$, $x'_n \leq supp'(h(g) \cup t(g))$.*

PROOF. If X is derivable from g, then $h(g) \subset X \subseteq h(g) \cup t(g)$; $GR(X) = \frac{supp''(X)}{supp'(X)} \leq \frac{x''_m}{x'_n}$ holds because $supp''(X) \leq supp''(h(g)) \leq x''_m$, $supp'(X) \geq supp'(h(g) \cup t(g)) \geq x'_n$. $\square$

x''_m is got from $usupp(g)$; if $supp'(h(g) \cup t(g))$ is known, we can immediately get $urate(g)$; otherwise, we compute x'_n according to the observation below.

OBSERVATION 3. *Given a dataset D and a group g, where $supp(h(g)) > 0$ and $supp(h(g) \cup t(g))$ is not available, the lower bound of $supp(h(g) \cup t(g))$ can be computed as follows: (1) if $\exists X \subseteq t(g)$ such that $supp(X) = 0$, $supp(h(g) \cup t(g)) = 0$; (2) $supp(h(g_p) \cup t(g_p)) \leq supp(h(g) \cup t(g))$; (3) $supp(h(g)) - \sum_{i \in t(g)}(supp(h(g_p) - supp(h(g_p) \cup \{i\})) \leq supp(h(g) \cup t(g))$; (4) $supp(h(g')) - \sum_{i \in t(g)}(supp(h(g')) - supp(h(g') \cup \{i\})) \leq supp(h(g) \cup t(g))$; (5) $supp(h(g') \cup t(g')) \leq supp(h(g) \cup t(g))$, where g_p is the parent of g, and g' is a group satisfying $h(g) = h(g')$, $t(g) \subset t(g')$.*

(1) is obvious. (2) follows directly from $h(g_p) \cup t(g_p) \supseteq h(g) \cup t(g)$. (3) and (4) follow from the support lower-bounding theorem of [3]. (5) follows from the fact that $h(g) \cup t(g) \subset h(g') \cup t(g')$. In EP mining, for a group g, if $\exists X \subseteq t(g)$ such that $sameOrg(X)$ or $topGR(X)$, from (1) we immediately know that $x'_n = 0$, otherwise we have to compute x'_n. To get a tighter x'_n, we compute several values from (2)-(5) and use the largest as x'_n. Note that (1), (2) and (3) are applicable to unprocessed, half-processed and completely processed groups, whereas (4) and (5) only apply to completely processed groups.

THEOREM 2. *Given a group g, $urateimp(g) = urate(g) - z$, where $z \leq max(GR(X)|X \subseteq h(g))$.*

PROOF. Given $X \subseteq h(g)$, X is a subset of any EP e derivable from g; so $rateimp(e) \leq GR(e) - z$. As $GR(e) \leq urate(g)$, we have $urateimp(g) = urate(g) - z$. $\square$

We apply Theorem 2 by letting $urateimp(g) = urate(g) - max(GR(X) \mid X \in E \wedge X \subseteq h(g))$, where E is the set of valid EPs known at the time.

3.3 Reordering tail items

We use heuristics to prune more effectively. Let g be a completely processed group that survives the pruning with both external and inherent constraints. Both inherent and external constraints provide ways for more pruning chances: (1) For all its child groups g_c to be generated, we know that $supp''(h(g_c)) \geq minsupp$ and $\neg topGR(h(g_c))$. If $\exists X \subseteq t(g_c)$ such that $sameOrg(X)$ or $topGR(X)$, then $supp'(X) = 0$; as $X \subseteq h(g_c) \cup t(g_c)$ $supp'(h(g_c) \cup t(g_c)) = 0$. By Theorem 1, $x''_m \geq supp''(h(g_c)) \geq minsupp$, $x'_n = 0$, and thus $urate(g_c) = \frac{x''_m}{x'_n} = \infty$. As $\neg topGR(h(g_c))$, $urateimp(g_c) = \infty$; g_c is not prunable. To avoid producing such g_c's, we put X (again $sameOrg(X)$ or $topGR(X)$) early in $t(g)$ so that more child groups will satisfy Observation 1 and be pruned on the fly. If there are several such X's, we order them by decreasing cardinality. (2) According to Theorem 1 and Theorem 2, based on an idea similar to Dense-Miner, we arrange the tail items in increasing order of $supp'(h(g) \cup \{i\})$ to produce more pruning chances. To apply both heuristics, as we always prune first with inherent constraints and pruning based on them is less costly, we always apply the inherent-constraint-based heuristics first.

4. CONSTRAINT-BASED EP MINER

Figure 2 gives a sketch of ConsEPMiner. Due to the same breadth-first search strategy over the SE-tree, the algorithm has the same skeleton as Dense-Miner: Given a level of surviving groups G, groups at the next level are generated by expanding surviving groups in G (`Prune-Gen-Next`). Valid EPs of G satisfying the given thresholds are accumulated in E (lines 6-7). The algorithm finishes when no new groups are generated (line 2). We will thus discuss below only what is specific to ConsEPMiner.

```
ConsEPMiner(background dataset D', target dataset D'')
;; T = {minsupp, minrate, minrateimp} is global
;; Return the EP set E of D'' satisfying T, E is global
1)  E ← φ; G ← Gen-Initial-Groups(D', D'')^(1);
2)  while G ≠ φ do
3)     scan D'' to process the groups in G;
4)     Prune-Groups(G)^(2);
5)     scan D' to process the groups in G;
6)     for each g ∈ G and i ∈ t(g) do
7)        E = E ∪ {h(g) ∪ {i} | h(g) ∪ {i} satisfies T};
8)     Prune-Groups(G)^(3);
9)     G ← Prune-Gen-Next(G)^(1);
10) return E;
```

Figure 2: Sketch of ConsEPMiner

A level of groups G is processed first in D'' and then in D', and this induces the 3-stage pruning of ConsEPMiner: as indicated by the superscripts of lines 1, 4, 8 and 9, pruning occurs in generating groups at a new level (stage 1), where `Prune-Groups` is called, and when groups are half- and completely-processed (stages 2 and 3). As can be seen from Figure 3, in `Prune-Groups`, we always try first to prune group tails with the less costly inherent constraints and the support threshold; only groups with the possibility of bounds > 0 are further pruned by computing the bounds of groups (`Rate-Imp-Bound-Prune`). Given a level of groups G, after the 3-stage pruning with inherent and external constraints, only groups with the possibility of deriving valid EPs are kept; only tail items that can produce such promising child groups are kept.

In `Prune-Gen-Next`, `Prune-Groups` is first called to prune newly generated groups and their tail items. As discussed in Observation 2, we derive the target support of the tail items of new groups by the same subset support constraint, reducing the database scanning time considerably.

Prune-Groups(groups at a level G)
;; Prune G, G is passed by reference
for each $g \in G$ **do**
  for each $i \in t(g)$ **do**
    if $sameOrg(h(g), i)^{(1)} \vee supp''(h(g) \cup \{i\}) < minsupp^{(2)}$
      $\vee sameSupp'(h(g), h(g) \cup \{i\})^{(3)} \vee topGR(h(g) \cup \{i\})^{(3)}$
      $\vee \exists e \in E \; s.t. \; (e \subset h(g) \cup \{i\} \wedge topGR(e))^{(3)}$
    remove i from $t(g)$;
  Rate-Imp-Bound-Prune(G, g)$^{(2,3)}$;

Figure 3: Function Prune-Groups

ConsEPMiner ensures that E consists of *all* EPs that are frequent and of large improvement with respect to their subsets that are valid EPs. E is already concise enough for most applications (e.g., classification). However, E may still contain some EPs that do not have large improvement with respect to their subsets that are not valid EPs. Post-processing is desirable to further remove such EPs from E, which can also be formulated as an SE-tree search problem [4].

5. EXPERIMENTS

We compare ConsEPMiner against two algorithms: EP-DenseMiner (as discussed in Section 1) and EP-Apriori, an EP mining algorithm based on the Apriori framework that also considers inherent constraints to strengthen pruning. All three algorithms were implemented in C++, with exactly the same implementation techniques.

We used the three algorithms to find EPs between data classes on many large dense UCI datasets and find that ConsEPMiner consistently outperforms EP-Apriori and EP-DenseMiner. We only report next the result on mining EPs of the Win class of Connect-4, with a 500MHz Pentium PC with 500M memory and with an execution time limit of 20,000 seconds. In Figure 4, the graph on the left shows the dominance of ConsEPMiner over EP-Apriori: While EP-Apriori can only find EPs at support greater than 88%, ConsEPMiner can successfully mine EPs at as low as 3%; at the support level where both algorithms succeed, ConsEPMiner is over 1000 times faster. While EP-DenseMiner is superior to EP-Apriori, the graph on the right clearly shows that ConsEPMiner outperforms EP-DenseMiner: EP-DenseMiner can only finish at higher support (5%); when

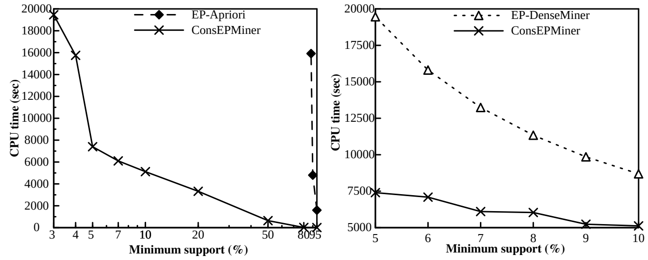

Figure 4: CPU Time: Connect-4($minrate$=2, $minimp$=0.01) ConsEPMiner vs. EP-Apriori vs. EP-DenseMiner

both algorithm succeed, ConsEPMiner is 2.63 times as fast as EP-DenseMiner.

6. CONCLUSIONS

We have presented and evaluated the algorithm ConsEPMiner for efficiently mining emerging patterns (EPs) from large dense datasets at low support. We have developed various techniques to use inherent and external constraints for effective pruning. The superior performance of ConsEPMiner shows that, pruning the search space directly with user-specified external constraints together with inherent constraints is a promising approach to inhibit candidate combinatorial explosion in data mining problems.

7. ACKNOWLEDGEMENTS

We would like to thank Roberto Bayardo for his insightful comments which helped improve the presentation.

8. REFERENCES

[1] R Agrawal & R Srikant. Fast algorithms for mining association rules. In *Proc. VLDB*, 1994.

[2] R J Bayardo. Brute-force mining of high-confidence classification rules. In *Proc. KDD*, 1997.

[3] R J Bayardo. Efficiently mining long patterns from databases. In *Proc. SIGMOD*, 1998.

[4] R J Bayardo, R Agrawal, & D Gunopulos. Constraint-based rule mining in large dense databases. In *Proc. ICDE*, 1999.

[5] G Dong & J Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. KDD*, 1999.

[6] G Dong, J Li, & X Zhang. Discovering emerging patterns in real datasets. In *Proc. IDC*, 1999.

[7] G Dong, X Zhang, L Wong, & J Li. CAEP: Classification by aggregating emerging patterns. In *Proc. Discovery Science*, 1999.

[8] L V S Lakshmanan, J Han R Ng, & A Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proc. SIGMOD*, 1999.

[9] R Ng, L V S Lakshmanan, J Han, & A Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. SIGMOD*, 1998.

[10] R Rymon. An SE-tree based characterization of the induction problem. In *Proc. ICML*, 1993.

[11] G I Webb. OPUS: An efficient admissible algorithm for unordered search. In *Journal of Artificial Intelligence Research*, 3:431-465, 1995.