Dynamic Re-ordering in Mining Top-k Productive Discriminative Patterns

Yoshitaka Kameya
Department of Information Engineering
Meijo University
Email: ykameya@meijo-u.ac.jp

Ken'ya Ito
Department of Information Engineering
Meijo University

***Abstract*—Discriminative patterns are the patterns that distinguish transactions in two different classes, one of which is typically of our particular interest. They are also known under the names of emerging patterns, contrast patterns, subgroup descriptions, and so on. In order to reduce the search space for top-k productive discriminative patterns, this paper proposes to re-order sibling patterns dynamically according to their quality. It is formally shown that the "sub-patterns first" property, which makes it easy to test the productivity of patterns, still holds for a re-ordered enumeration tree. Moreover, in an extensive experiment, we observed that the proposed method shows a stable performance in various settings, and reduces the search space drastically for some burdensome situations. It is also found that the proposed algorithm works well as an anytime algorithm.**

I. Introduction

Discriminative patterns are the patterns that distinguish transactions in two different classes, one of which is typically of our particular interest. For example, some may wish to find characteristic patterns in votes by Republicans for various key issues at United States House of Representatives. Thanks to the annotated class information, discriminative patterns tend to be more informative than frequent patterns, and can be a basis of precise rule-based classifiers [1]. Discriminative patterns are also known under the names of emerging patterns, contrast patterns, subgroup descriptions, and so on [2], [3]. Throughout the paper, we focus on mining itemset patterns.

In discriminative pattern mining, the top-k constraint [4] and branch-and-bound pruning have been exploited for mitigating the difficulty in handling the quality function violating anti-monotonicity [5]. Another difficulty, redundancy among patterns, has also been mitigated by introducing set-inclusion-based constraints among patterns. For example, under the productivity constraint [6], [7], [8], patterns to be output (e.g. $\{\mathsf{A},\mathsf{C},\mathsf{D}\}$) must be of higher quality than all of its sub-patterns ($\{\mathsf{A}\}$, $\{\mathsf{C}\}$, $\{\mathsf{D}\}$, $\{\mathsf{A},\mathsf{C}\}$, $\{\mathsf{A},\mathsf{D}\}$ and $\{\mathsf{C},\mathsf{D}\}$).

One convenient fact about the productivity constraint is that it is easily tested in a depth-first search over a *suffix enumeration tree* [9]. To illustrate, Fig. 1 (a) and (b) present two enumeration trees for patterns made up of items A, B, C and D. Fig. 1 (a) (resp. Fig. 1 (b)) is called a prefix (resp. suffix) enumeration tree because the parent of each pattern $\boldsymbol{x}$ is the immediate prefix (resp. suffix) of $\boldsymbol{x}$. In both trees, we typically insert eligible single items into each parent pattern following some pre-defined or *static* total order $\prec$. For example, letting $\prec$ be the alphabetical order, in Fig. 1 (b), we prepend A, B and C (which precede D w.r.t. $\prec$) into a parent $\{\mathsf{D}\}$ and obtain sibling patterns $\{\mathsf{A},\mathsf{D}\}$, $\{\mathsf{B},\mathsf{D}\}$ and $\{\mathsf{C},\mathsf{D}\}$ in turn. FP-growth [10] is known to run over a suffix enumeration tree [11], and what is less known is that, *in a depth-first and left-to-right traversal over a suffix enumeration tree, at the moment we visit a pattern $\boldsymbol{x}$, all the sub-patterns of $\boldsymbol{x}$ have been visited.* We call this property the "sub-patterns first" property. For example, in Fig. 1 (b), $\{\mathsf{A},\mathsf{C},\mathsf{D}\}$ is visited after $\{\mathsf{A}\}$, $\{\mathsf{C}\}$, $\{\mathsf{D}\}$, $\{\mathsf{A},\mathsf{C}\}$, $\{\mathsf{A},\mathsf{D}\}$ and $\{\mathsf{C},\mathsf{D}\}$ have been visited, but this is not the case with Fig. 1 (a). Since a pattern's quality is usually evaluated when the pattern is visited, the "sub-patterns first" property enables us to compare the quality of the pattern $\boldsymbol{x}$ we are visiting with the qualities of all the sub-patterns of $\boldsymbol{x}$, which have already been evaluated, and then to judge *on the fly* whether $\boldsymbol{x}$ is productive. To the best of our knowledge, the "sub-patterns first" property was introduced first in selecting frequent minimal generators [12], but has not been certified in a formal sense.

Suffix enumeration trees are also beneficial when combined with the top-k constraint [4]. In top-k mining, we often prepare a candidate list of size k, into which a pattern $\boldsymbol{x}$ of higher quality than the k-th pattern $\boldsymbol{z}$ in the list will be added. Furthermore, a pattern $\boldsymbol{x}$ whose descendants cannot have higher quality than $\boldsymbol{z}$ is safely pruned [9]. Here, let us see the suffix enumeration tree in Fig. 1 (b) again and suppose that items A, B, C and D have higher quality in this order (an item is often seen as a pattern that contains only the item itself). One may find that, in this suffix enumeration tree, we visit patterns containing items of higher quality (such as $\{\mathsf{A},\mathsf{B}\}$) earlier, and hence in top-k mining, the k-th pattern's quality tends to be raised more quickly. This phenomenon makes effective the pruning based on the k-th pattern's quality, and eventually reduces the search space.

In this paper, for further reduction of the search space for top-k productive discriminative patterns, we propose to re-order sibling patterns *dynamically* according to their quality (a detailed illustration will be given in Section II). A formal contribution of this paper is to show that the "sub-patterns first" property still holds for a re-ordered version of suffix enumeration trees, and therefore it is still easy to examine the productivity constraint. Moreover, in an extensive experiment, we observed that the proposed method shows a stable

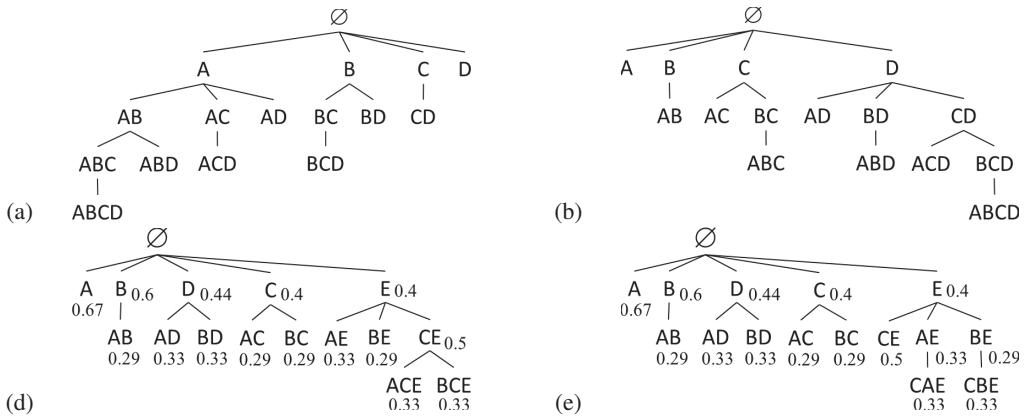

Fig. 1. (a) A typical prefix enumeration tree, (b) a typical suffix enumeration tree, (c) an exemplar transactional database, (d) a suffix enumeration tree for finding discriminative patterns in the exemplar database, and (e) its dynamically-reordered version. Patterns are considered to be visited in a depth-first and left-to-right manner. For space limitation, each pattern $\{x, y, z, \ldots\}$ in enumeration trees is abbreviated as $xyz\cdots$, where x, y, z, ... are items.

performance in various settings, and reduces the search space drastically in some burdensome situations. It is also found that the proposed algorithm works well as an anytime algorithm, i.e. it can output patterns of satisfactory quality even after an early interruption by the user.

This paper is structured as follows. In Section II, we show an example that illustrates our proposed method. Then, Section III briefly introduces several background notions and notations used in the paper. Section IV describes the details of our proposed method, and Section V presents the results of our experiments. Lastly, Section VI concludes the paper, mentioning some related work.

II. An Illustrative Example

Let us consider a database containing five transactions in class $+$ and five transactions in class $-$ in Fig. 1 (c). Also suppose that class $+$ is of our interest. Then, we have a suffix enumeration tree in Fig. 1 (d),[1] where items to be prepended are chosen firstly in the descending order of their quality, say, F-score, and secondly in the alphabetical order (i.e. in order of A, B, D, C and E). For the class c of interest, the F-score of a pattern $\boldsymbol{x}$ is defined as the harmonic mean of (positive) support $p(\boldsymbol{x} \mid c)$ and confidence $p(c \mid \boldsymbol{x})$. For $c = +$ and $\boldsymbol{x} = \{\mathsf{A}\}$, we have $p(\boldsymbol{x} \mid c) = 3/5 = 0.6$ and $p(c \mid \boldsymbol{x}) = 3/4 = 0.75$, and thus $\boldsymbol{x}$'s F-score is computed as $2 \times 0.6 \times 0.75/(0.6 + 0.75) = 0.67$. During the search, each pattern's F-score is compared with the F-scores of all its sub-patterns, and finally, only patterns $\{\mathsf{A}\}$, $\{\mathsf{B}\}$, $\{\mathsf{C}, \mathsf{E}\}$, $\{\mathsf{D}\}$, $\{\mathsf{C}\}$ and $\{\mathsf{E}\}$ are output as productive ones. Now one may find that the combination of C and E has something meaningful for class $+$.

As said before, we propose to re-order sibling patterns (or equivalently, items to be prepended) dynamically according to their quality. The re-ordered version of Fig. 1 (d) is given in Fig. 1 (e). The difference between them lies in the subtree rooted by pattern $\{\mathsf{E}\}$, where pattern $\{\mathsf{C}, \mathsf{E}\}$ is visited earlier than $\{\mathsf{A}, \mathsf{E}\}$ and $\{\mathsf{B}, \mathsf{E}\}$. This is because the F-score of $\{\mathsf{C}, \mathsf{E}\}$ is evaluated as higher than the F-scores of $\{\mathsf{A}, \mathsf{E}\}$ and $\{\mathsf{B}, \mathsf{E}\}$. It is obvious that the re-ordered version can collect patterns of higher quality earlier, and is beneficial for top-k mining. Interestingly, whereas the parent-child relations among patterns below $\{\mathsf{E}\}$ have been changed significantly, the "sub-patterns first" property is still kept. Later, we formally show that the "sub-patterns first" property is guaranteed even with dynamic re-ordering of sibling patterns, as in this example.

[1]The patterns that do not occur in the transactions in class $+$ do not appear in the enumeration tree.

III. Background

A. Preliminaries

We first consider a dataset $\mathcal{D} = \{t_1, t_2, \ldots, t_N\}$, where t_i $(1 \leq i \leq N)$ is a set of items called a transaction. Each transaction belongs to one of pre-defined classes $\mathcal{C}$, and let c_i be the class of transaction t_i. A pattern $\boldsymbol{x}$ is a subset of items appearing in $\mathcal{D}$. Items are usually referred to by variables x, y, z, and so on, while concrete items are named A, B, C, and so on. We will interchangeably denote a pattern as a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, as a set $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$, or as a conjunction $\boldsymbol{x} = (x_1 \wedge x_2 \wedge \ldots \wedge x_n)$ of items. An item x is often considered as a singleton pattern $\{x\}$.

We further define $\mathcal{D}_c = \{t_i \mid c_i = c, 1 \leq i \leq N\}$, $\mathcal{D}(\boldsymbol{x}) = \{t_i \mid \boldsymbol{x} \subseteq t_i, 1 \leq i \leq N\}$ and $\mathcal{D}_c(\boldsymbol{x}) = \{t_i \mid c_i = c, \boldsymbol{x} \subseteq t_i, 1 \leq i \leq N\}$, where c is the class of interest. A symbol $\neg$ is used for negation, e.g. $\mathcal{D}_{\neg c} = \mathcal{D} \setminus \mathcal{D}_c$, $\mathcal{D}_{\neg c}(\boldsymbol{x}) = \mathcal{D}(\boldsymbol{x}) \setminus \mathcal{D}_c(\boldsymbol{x})$ and so on. Probabilities are computed from $\mathcal{D}$. A joint probability $p(c, \boldsymbol{x})$ is obtained as $|\mathcal{D}_c(\boldsymbol{x})|/N$. Similarly we have $p(\neg c, \boldsymbol{x}) = |\mathcal{D}_{\neg c}(\boldsymbol{x})|/N$ and so on. Marginal and conditional probabilities are computed in a standard way, e.g. $p(\boldsymbol{x}) = p(c, \boldsymbol{x}) + p(\neg c, \boldsymbol{x})$ and $p(c \mid \boldsymbol{x}) = p(c, \boldsymbol{x})/p(\boldsymbol{x})$. We call conditional probabilities $p(\boldsymbol{x} \mid c)$, $p(\boldsymbol{x} \mid \neg c)$ and $p(c \mid \boldsymbol{x})$ positive support, negative support and confidence, respectively.

B. Dual-monotonicity

The quality of a pattern $\boldsymbol{x}$ for class c is written as $R_c(\boldsymbol{x})$, and most of popular quality functions are defined using

Algorithm 1 GROW(T_0, $\boldsymbol{x}_0$)

Require: T_0: the current FP-tree, $\boldsymbol{x}_0$: the current pattern
1: $H_0 :=$ the header table associated with T_0
2: **for all** x in the key items of H_0 enumerated in the *ascending* order w.r.t. $\prec$ **do** ▷ Branching by an item preceding all items in $\boldsymbol{x}_0$
3: $\boldsymbol{x} := \{x\} \cup \boldsymbol{x}_0$ ▷ A new pattern we are visiting
4: $\boldsymbol{z} :=$ the pattern having the k-th highest quality in L
5: Construct an FP-tree T conditioned on x from T_0, computing $R_c(\boldsymbol{x})$, $\overline{R}_c(\boldsymbol{x})$ and $p(c \mid \boldsymbol{x})$ from the statistics stored in T, and removing every item y such that $\overline{R}_c(\{y\} \cup \boldsymbol{x}) < R_c(\boldsymbol{z})$ ▷ Constructing a new FP-tree with branch-and-bound pruning based on $\boldsymbol{z}$
6: **if** $p(c \mid \boldsymbol{x}) \geq p(c)$ and $\boldsymbol{x}$ is *not* weaker than any pattern in L **then** ▷ Adding $\boldsymbol{x}$ as a top-k productive pattern if it is qualified
7: Insert $\boldsymbol{x}$ into L following the descending order of R_c
8: Remove the patterns of the $(k+1)$-th highest quality (if any) from L
9: **end if**
10: GROW(T, $\boldsymbol{x}$) **if** $\boldsymbol{x}$ is *not* prunably weaker than any pattern in L ▷ Recursion or branch-and-bound pruning based on productivity
11: **end for**

positive support $p(\boldsymbol{x} \mid c)$ and negative support $p(\boldsymbol{x} \mid \neg c)$ [13]. Hereafter we regard the quality of an item x for c as $R_c(\{x\})$. As an instance of R_c, throughout the paper, we use *F-score* $\mathrm{F}_c(\boldsymbol{x}) = 2p(c \mid \boldsymbol{x})p(\boldsymbol{x} \mid c)/(p(c \mid \boldsymbol{x}) + p(\boldsymbol{x} \mid c))$. Since we seek for the patterns characterizing c, we focus on the patterns $\boldsymbol{x}$ such that $p(\boldsymbol{x} \mid c) \geq p(\boldsymbol{x} \mid \neg c)$ or equivalently $p(c \mid \boldsymbol{x}) \geq p(c)$. Recently, a relaxed condition called *dual-monotonicity* was introduced in [13]. That is, a quality function R_c for a class c is dual-monotonic iff, for any pattern $\boldsymbol{x}$, $R_c(\boldsymbol{x})$ is monotonically increasing w.r.t. $p(\boldsymbol{x} \mid c)$ and monotonically decreasing w.r.t. $p(\boldsymbol{x} \mid \neg c)$ wherever $p(\boldsymbol{x} \mid c) \geq p(\boldsymbol{x} \mid \neg c)$. Several popular quality functions including F-score, χ^2, information gain and support difference are all dual-monotonic.

C. Branch-and-bound Pruning in Top-k Mining

Suppose that we perform a branch-and-bound search for top-k patterns under a dual-monotonic quality function R_c, and consider an anti-monotonic upper bound $\overline{R}_c(\boldsymbol{x})$ of $R_c(\boldsymbol{x})$ for a pattern $\boldsymbol{x}$. Then, if $\overline{R}_c(\boldsymbol{x}) < R_c(\boldsymbol{z})$, where $\boldsymbol{z}$ is the pattern of the k-th highest quality at the moment, we can safely prune the subtree rooted by $\boldsymbol{x}$ in the enumeration tree. This pruning exploits the anti-monotonicity of $\overline{R}_c$ w.r.t. pattern-inclusion, which guarantees $R_c(\boldsymbol{x}') \leq \overline{R}_c(\boldsymbol{x}') \leq \overline{R}_c(\boldsymbol{x}) < R_c(\boldsymbol{z})$ for any super-pattern $\boldsymbol{x}'$ of $\boldsymbol{x}$. If R_c is dual-monotonic, an anti-monotonic upper bound $\overline{R}_c(\boldsymbol{x})$ is obtained by substituting $p(\boldsymbol{x} \mid \neg c) := 0$ into the definition of $R_c(\boldsymbol{x})$ [13].

D. The Productivity Constraint

Here we will formally give a definition of the productivity constraint, whose previous versions are given together with confidence as a quality function [6], [7]. To be specific, for a class c of interest and a pair of patterns $\boldsymbol{x}$ and $\boldsymbol{x}'$, we say that $\boldsymbol{x}$ is *weaker than* $\boldsymbol{x}'$ iff $\boldsymbol{x} \supset \boldsymbol{x}'$ and $R_c(\boldsymbol{x}) \leq R_c(\boldsymbol{x}')$. A pattern $\boldsymbol{x}$ is then said to be *productive* iff $\boldsymbol{x}$ is not weaker than any pattern.

Moreover, we can conduct an aggressive pruning based on an extended notion of weakness. First, we say that a pattern $\boldsymbol{x}$ is *prunably weaker than* $\boldsymbol{x}'$ iff $\boldsymbol{x} \supset \boldsymbol{x}'$ and $\overline{R}_c(\boldsymbol{x}) \leq R_c(\boldsymbol{x}')$. Then, if $\boldsymbol{x}$ is prunably weaker than some pattern $\boldsymbol{x}'$ in the current top-k candidates, any super-pattern of $\boldsymbol{x}$ is also weaker than $\boldsymbol{x}'$, and so we can safely prune the subtree rooted by $\boldsymbol{x}$.

E. FP-growth for Discriminative Pattern Mining

Based on the notions above, now we introduce a variant of the FP-growth algorithm [10] that finds top-k productive patterns highly relevant to the class c of our interest. This variant conducts a depth-first and left-to-right search over a suffix enumeration tree and branch-and-bound pruning. To be more specific, we introduce a list L that stores the candidates for top-k patterns, construct an initial FP-tree T_{init} from the input dataset $\mathcal{D}$, prepare an initial, empty pattern $\emptyset$, and run GROW(T_{init}, $\emptyset$), which is shown in Algorithm 1.

Remark here that the order $\prec$ among items is referred to as static information. To make branch-and-bound pruning effective, under $\prec$, we usually order items firstly by their quality and secondly by their alphabetical order. For instance, the items in the original transactions in $\mathcal{D}$ and in the conditional transactions, that form a new FP-tree, are always placed in the ascending order w.r.t. $\prec$. Also, in Line 2 of GROW, we pick up items to be prepended according to the ascending order w.r.t. $\prec$. As a result, a new FP-tree T conditioned on x (Line 5) and its header table contain only the items that precede all items in $\boldsymbol{x} = \{x\} \cup \boldsymbol{x}_0$ w.r.t. $\prec$, and the enumeration tree followed by GROW turns to be a suffix enumeration tree like Fig. 1 (b). In our proposed method described next, instead of the *static* order $\prec$, we introduce a *dynamic* order $\prec_{\boldsymbol{x}}$ among items conditioned on the pattern $\boldsymbol{x}$ currently we are visiting.

IV. THE PROPOSED METHOD

To accelerate GROW above, we propose to re-order sibling patterns dynamically according their quality. The resulting algorithm is GROW-AND-REORDER in Algorithm 2. Here, we introduce an order $\prec_{\boldsymbol{x}}$ among items conditioned on $\boldsymbol{x}$. For two items x and y that are not the members of $\boldsymbol{x}$, "$x \prec_{\boldsymbol{x}} y$" states that $R_c(\{x\} \cup \boldsymbol{x}) > R_c(\{y\} \cup \boldsymbol{x})$, or x precedes y alphabetically when $R_c(\{x\} \cup \boldsymbol{x}) = R_c(\{y\} \cup \boldsymbol{x})$. Such a conditional order makes promising items come earlier, and is used in enumerating the key items in the header table (Line 2), and in sorting items in conditional transactions (Line 5). Finally the new conditional order is passed into the recursive call (Line 10). At the top level, similarly to GROW, we run GROW-AND-REORDER(T_{init}, $\emptyset$, $\prec$), where the third argument $\prec$ is the order among items which is statically referred to in GROW, but just works as an initial order $\prec_{\emptyset}$ here.

Algorithm 2 GROW-AND-REORDER(T_0, $\boldsymbol{x}_0$, $\prec_{\boldsymbol{x}_0}$)

Require: T_0: the current FP-tree, $\boldsymbol{x}_0$: the current pattern, $\prec_{\boldsymbol{x}_0}$: the order among items conditioned on $\boldsymbol{x}_0$

$H_0 :=$ the header table associated with T_0
for all x in the key items of H_0 enumerated in the *ascending* order w.r.t. $\prec_{\boldsymbol{x}_0}$ **do**
  $\boldsymbol{x} := \{x\} \cup \boldsymbol{x}_0$ and let $\prec_{\boldsymbol{x}}$ be the order conditioned on $\boldsymbol{x}$
  $\boldsymbol{z} :=$ the pattern having the k-th highest quality in L
  Construct an FP-tree T conditioned on x from T_0, removing every item y such that $\overline{R}_c(\{y\} \cup \boldsymbol{x}) < R_c(\boldsymbol{z})$, computing $R_c(\boldsymbol{x})$, $\overline{R}_c(\boldsymbol{x})$ and $p(c \mid \boldsymbol{x})$ from the statistics stored in T, and placing items in each conditional transactions in the ascending order w.r.t. $\prec_{\boldsymbol{x}}$
  if $p(c \mid \boldsymbol{x}) \geq p(c)$ and $\boldsymbol{x}$ is *not* weaker than any pattern in L **then**
    Insert $\boldsymbol{x}$ into L following the descending order of R_c
    Remove the patterns of the $(k+1)$-th highest quality (if any) from L
  end if
  GROW-AND-REORDER(T, $\boldsymbol{x}$, $\prec_{\boldsymbol{x}}$) **if** $\boldsymbol{x}$ is *not* prunably weaker than any pattern in L
end for

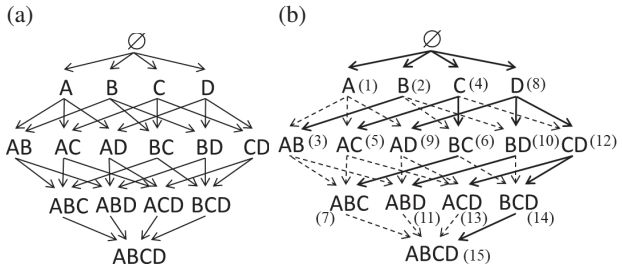

Fig. 2. (a) A Hasse diagram depicted as a directed acyclic graph, and (b) the trace of a topological sorting over the Hasse diagram, in which solid lines indicate the caller-callee relations in recursive calls of VISIT, and the number associated with each pattern is the position in the output sequence.

To make dynamic re-ordering work efficiently with the productivity constraint, the "sub-patterns first" property introduced before must be guaranteed in GROW-AND-REORDER. As a starting point, using the notion of topological sorting [14], we show that the "sub-patterns first" property holds in GROW, which only refers to a static order $\prec$ among items.

First, let us consider a Hasse diagram in the form of a directed acyclic graph, having edges from patterns to their immediate super-patterns (Fig. 2 (a)). Then, by a traversal over the Hasse diagram from an empty pattern, a topological sorting generates a sequence of patterns in which any pattern cannot be a sub-pattern of its preceding patterns. A key observation here is that *the "sub-patterns first" property holds for the enumeration tree of a depth-first search if there exists a topological sorting over a Hasse diagram that generates a sequence in the same order as the visiting order of the search*.

To be specific, suppose $\mathsf{A} \prec \mathsf{B} \prec \mathsf{C} \prec \mathsf{D}$, and consider a topological sorting over the Hasse diagram in Fig. 2 (a), which is realized by Algorithms 3 and 4. In Algorithm 4, $head(\boldsymbol{x})$ is defined as a set of items that precede all items in $\boldsymbol{x}$ w.r.t. $\prec$, and by PUSH(S, $\boldsymbol{x}$) we prepend $\boldsymbol{x}$ into a sequence S. Then, the trace of VISIT's *right-to-left* recursive calls are highlighted in Fig. 2 (b), together with the positions of the patterns in the output sequence S. Since the order of patterns in S is exactly the same as the visiting order in GROW's depth-first and *left-to-right* traversal over the suffix enumeration tree in Fig. 1 (b), we can say that there certainly exists a topological sorting that

Algorithm 3 TOPOLOGICALLY-SORT

$S :=$ an empty sequence
VISIT($\emptyset$, S)
return S

Algorithm 4 VISIT($\boldsymbol{x}$, S)

Require: $\boldsymbol{x}$: the current pattern, S: a sequence of patterns

for all x in $head(\boldsymbol{x})$ in the *descending* order w.r.t. $\prec$ **do**
  VISIT($\{x\} \cup \boldsymbol{x}$, S)
end for
PUSH(S, $\boldsymbol{x}$)

guarantees the "sub-patterns first" property in GROW.

Let us further see this formally, from the order of recursive calls and the way of inserting patterns into S. We first note that, in Algorithm 4, VISIT($\boldsymbol{x}$, S) calls VISIT($\{x_m\} \cup \boldsymbol{x}$, S), VISIT($\{x_{m-1}\} \cup \boldsymbol{x}$, S), ..., VISIT($\{x_1\} \cup \boldsymbol{x}$, S) in turn, and then prepends $\boldsymbol{x}$ into S, where $x_1, x_2, \ldots, x_m$ are the items in $head(\boldsymbol{x})$ and $x_i \prec x_j$ if $i < j$. Let $\sigma(\boldsymbol{x})$ be the sequence of the patterns prepended into S during the call of VISIT($\boldsymbol{x}$, S), and see that $\sigma(\boldsymbol{x})$ is a concatenation of $\boldsymbol{x}$, $\sigma(\{x_1\} \cup \boldsymbol{x})$, ..., $\sigma(\{x_{m-1}\} \cup \boldsymbol{x})$ and $\sigma(\{x_m\} \cup \boldsymbol{x})$. Also note that $head(\boldsymbol{x})$ is identical to the set of key items in the header table in GROW($\cdot$, $\boldsymbol{x}$). Then, we see that the order among patterns in $\sigma(\boldsymbol{x})$ is exactly the same as the visiting order in the subsequent recursive calls of GROW. Branch-and-bound pruning does not affect the visiting order, and we can say that the "sub-patterns first" property is generally guaranteed in GROW.

Next, for the case with dynamic re-ordering, we can introduce Algorithms 5 and 6 as another routine for topological sorting. Here, $head^*(\boldsymbol{x})$ is defined as a set of items that precede all items in $\boldsymbol{x}$ w.r.t. $\prec_{\boldsymbol{x}'}$ for any suffix $\boldsymbol{x}'$ of $\boldsymbol{x}$. Then, a similar discussion is also possible for GROW-AND-REORDER, and consequently, the "sub-patterns first" property is guaranteed in GROW-AND-REORDER.

V. EXPERIMENTAL RESULTS

We conducted an extensive experiment in order to measure the effect of dynamic re-ordering in reducing the search space. The target datasets are the datasets available from http://dtai.cs.kuleuven.be/CP4IM/datasets/. The statistics on the datasets are summarized in Table I. With these datasets, we compare the

Algorithm 5 TOPOLOGICALLY-SORT-AND-REORDER

1: $S :=$ an empty sequence
2: VISIT-AND-REORDER($\emptyset$, S, $\prec$)
3: **return** S

Algorithm 6 VISIT-AND-REORDER($\boldsymbol{x}_0$, S, $\prec_{\boldsymbol{x}_0}$)

Require: $\boldsymbol{x}_0$: the current pattern, S: a sequence of patterns, $\prec_{\boldsymbol{x}_0}$: the order among items conditioned on $\boldsymbol{x}_0$
1: **for all** x in $head^*(\boldsymbol{x}_0)$ in the *descending* order w.r.t. $\prec_{\boldsymbol{x}_0}$ **do**
2: $\quad\boldsymbol{x} := \{x\} \cup \boldsymbol{x}_0$ and let $\prec_{\boldsymbol{x}}$ be the order conditioned on $\boldsymbol{x}$
3: $\quad$VISIT-AND-REORDER($\boldsymbol{x}$, S, $\prec_{\boldsymbol{x}}$)
4: **end for**
5: PUSH(S, $\boldsymbol{x}_0$)

search space (the number of visited patterns in the enumeration tree) and the running time among a variant of FP-growth with static ordering based on the quality of items (named "Static"), the one with static random ordering ("Random"), and the one with dynamic re-ordering ("Dynamic"). In static random ordering, we first decide the order among items at random, and always refer to it during the search. In the experiment, we used F-score as the quality function of a pattern. The number k of output patterns was chosen from 1, 10 and 50. We tried 30 ways of static random ordering for each dataset and each setting. All runs were conducted on Intel Core i7 3.6GHz and we terminated the runs that had exceeded 15 minutes.

In addition, it would be interesting to see whether the proposed method (GROW-AND-REORDER) can work as an anytime algorithm, i.e. whether it can output patterns of satisfactory quality even after an early interruption by the user. To answer this question, in a retrospective fashion, we identified the true top-k pattern $\boldsymbol{x}_{\rm last}$ found lastly, and recorded the *effective* number of visited patterns, i.e. the number of patterns visited until $\boldsymbol{x}_{\rm last}$ has been visited.

Here we add some notes on our implementation. GROW in Algorithm 1 is in fact a simplified version of the original one presented in [9]. The original version introduces some further improvements (such as the translation of the upper bound of quality into the minimum support threshold) which may affect the running time, though it traverses the same search space. In our Java implementation, the variants of FP-growth above inherit all the improvements from the original version.

The results of the runs with $k = 1$, $k = 10$ and $k = 50$ are shown in the top, the center and the bottom of Table II, respectively.[2] In this table, the figures captioned by "#Visited patterns (entire)" measure the entire search space traversed. Besides, the figures captioned by "#Visited patterns (effective)" are the effective numbers of visited patterns, which are introduced above, and measure the search space that suffices to find true top-k patterns. The rightmost part of Table II shows the time required for traversing the entire search space. The captions "Static," "Dynamic" and "Random" are the names of the variants of FP-growth performed, and the column "Ratio"

[2]For the audiology dataset, no run finished within 15 minutes. For the hypothyroid dataset, the runs with $k = 50$ did not finish within 15 minutes. So, in Table II, the statistics on these runs are omitted.

TABLE I
STATISTICS ON THE DATASETS USED IN THE EXPERIMENT.

Dataset	#Trans.	#Items	Dataset	#Trans.	#Items
anneal	812	93	lymph	148	68
audiology	216	148	mushroom	8,124	119
australian-credit	653	125	primary-tumor	336	31
german-credit	1,000	112	soybean	630	50
heart-cleveland	296	95	splice-1	3,190	287
hepatitis	137	68	tic-tac-toe	958	27
hypothyroid	3,247	88	vote	435	48
kr-vs-kp	3,196	73	zoo-1	101	36

presents the reduction ratios of the search space of "Dynamic." to that of "Static." Some of the figures in the table take an exponential form, i.e. "$f\mathrm{E}\pm i$" stands for $f \times 10^{\pm i}$.

From the results in Table II, we first see that the variant "Dynamic" shows a stable performance, even in burdensome situations where the search space is inherently large, e.g. in the runs for german-credit and hepatitis with $k = 50$. On the other hand, as for the running time, "Dynamic" often runs slightly slower than "Static" in lightweight situations. This is presumably because there is some overhead in dynamic re-ordering, such as sorting items in conditional transactions. Of course, such an overhead seems ignorable in lightweight situations. Lastly, by comparing the results in "#Visited patterns (entire)" and "#Visited patterns (effective)," we see that all of true top-k patterns are often found much earlier than the entire search finishes, and again, for burdensome situations, dynamic re-ordering contributes to deriving a better anytime mining algorithm.

VI. CONCLUDING REMARKS

In this paper, for reducing the search space for top-k productive discriminative patterns, we proposed to re-order sibling patterns dynamically according to their quality, and confirmed in an extensive experiment that the search space can surely be reduced by dynamic re-ordering, and the resulting algorithm can work well as an anytime algorithm.

Nowadays the idea of dynamic re-ordering in branch-and-bound search seems not novel itself. For instance, in the literature related to discriminative pattern mining, dynamic re-ordering was adopted in the OPUS [15] algorithm family for subset selection problems including feature selection, and in SD-Map* [16] for subgroup discovery for continuous target concepts. This paper, however, further introduced the productivity constraint in order to achieve non-redundancy among patterns, and showed formally that the "sub-patterns first" property, which makes it easy to test productivity, is guaranteed even with dynamic re-ordering. In addition, guaranteeing the "sub-patterns first" property using the notion of topological sorting would also bring benefits with well-foundedness to the cases that require some minimality of patterns [12], or to the cases in which a domain-dependent concept hierarchy is exploited for producing generalized patterns [17].

REFERENCES

[1] F. Thabtah, "A review of associative classification mining," *Knowledge Engineering Review*, vol. 22, no. 1, pp. 37–65, 2007.

TABLE II
COMPARISON ON SEARCH PERFORMANCE (TOP: $k = 1$, CENTER: $k = 10$, BOTTOM: $k = 50$).

Dataset	#Visited patterns (entire)				#Visited patterns (effective)				Running time (in seconds)			
	Static	Dynamic	Random	Ratio	Static	Dynamic	Random	Ratio	Static	Dynamic	Random	Ratio
anneal	2.4E+5	2.4E+5	2.5E+5 (2.1E+4)	0.00	2.5E+1	2.5E+1	5.9E+4 (6.4E+4)	0.00	1.11	1.30	1.15 (0.30)	−0.17
australian-credit	5.1E+3	5.1E+3	1.1E+4 (1.0E+4)	0.00	1.0E+0	1.0E+0	6.6E+3 (1.2E+4)	0.00	0.49	0.64	0.64 (0.24)	−0.29
german-credit	3.4E+2	3.4E+2	3.6E+2 (1.6E+1)	0.00	1.0E+0	1.0E+0	1.4E+2 (9.8E+1)	0.00	0.40	0.40	0.44 (0.07)	0.01
heart-cleveland	5.7E+3	5.7E+3	7.1E+3 (1.5E+3)	0.00	5.7E+2	5.3E+2	3.8E+3 (3.0E+3)	0.06	0.45	0.45	0.61 (0.09)	−0.01
hepatitis	8.0E+1	8.0E+1	9.0E+1 (5.5E+0)	0.00	1.0E+0	1.0E+0	5.3E+1 (2.1E+1)	0.00	0.06	0.07	0.08 (0.00)	−0.07
hypothyroid	1.2E+3	1.2E+3	2.5E+3 (2.2E+3)	0.00	1.0E+0	1.0E+0	1.5E+3 (2.5E+3)	0.00	0.73	0.76	0.77 (0.11)	−0.03
kr-vs-kp	2.0E+5	2.0E+5	2.6E+5 (1.1E+5)	0.00	9.9E+2	9.9E+2	1.4E+5 (1.6E+5)	0.00	0.86	1.52	1.71 (0.47)	−0.76
lymph	1.1E+4	1.1E+4	1.2E+4 (2.6E+2)	0.00	1.0E+0	1.0E+0	9.5E+2 (2.1E+3)	0.00	0.44	0.48	0.44 (0.03)	−0.08
mushroom	1.2E+2	1.2E+2	1.4E+2 (2.0E+1)	0.00	1.0E+0	1.0E+0	6.4E+1 (4.8E+1)	0.00	0.21	0.21	0.44 (0.09)	0.01
primary-tumor	8.8E+2	8.8E+2	1.1E+3 (4.6E+2)	0.00	1.8E+1	1.8E+1	4.7E+2 (6.7E+2)	0.00	0.09	0.10	0.11 (0.02)	−0.13
soybean	4.3E+3	4.3E+3	5.1E+3 (6.0E+2)	0.00	2.1E+2	2.1E+2	2.7E+3 (1.6E+3)	0.00	0.21	0.23	0.24 (0.02)	−0.09
splice-1	2.5E+2	2.5E+2	2.5E+2 (2.5E+0)	0.00	1.0E+0	1.0E+0	1.3E+2 (7.7E+1)	0.00	0.65	0.65	0.66 (0.01)	0.00
tic-tac-toe	2.7E+1	2.7E+1	2.8E+1 (1.0E+0)	0.00	1.0E+0	1.0E+0	1.2E+1 (7.4E+0)	0.00	0.05	0.04	0.05 (0.00)	0.17
vote	4.8E+1	4.8E+1	5.1E+1 (2.0E+0)	0.00	1.0E+0	1.0E+0	3.0E+1 (1.4E+1)	0.00	0.05	0.05	0.05 (0.00)	0.06
zoo-1	5.4E+1	5.4E+1	7.2E+1 (2.9E+1)	0.00	1.0E+0	1.0E+0	4.0E+1 (4.1E+1)	0.00	0.03	0.02	0.03 (0.00)	0.18

Dataset	#Visited patterns (entire)				#Visited patterns (effective)				Running time (in seconds)			
	Static	Dynamic	Random	Ratio	Static	Dynamic	Random	Ratio	Static	Dynamic	Random	Ratio
anneal	3.9E+5	3.9E+5	4.0E+5 (2.4E+4)	0.00	2.7E+2	2.7E+2	1.7E+5 (1.1E+5)	0.03	1.58	1.08	1.34 (0.27)	0.32
australian-credit	7.1E+4	6.7E+4	8.5E+4 (4.9E+4)	0.06	2.6E+3	1.8E+3	5.6E+4 (4.8E+4)	0.31	0.96	0.51	1.18 (0.38)	0.47
german-credit	9.9E+2	8.4E+2	3.0E+3 (9.9E+2)	0.15	6.0E+2	5.4E+2	2.7E+3 (1.0E+3)	0.09	0.43	0.46	0.60 (0.16)	−0.06
heart-cleveland	1.2E+4	1.2E+4	1.4E+4 (2.1E+3)	0.00	7.6E+2	6.9E+2	8.6E+3 (3.9E+3)	0.09	0.61	0.60	0.74 (0.10)	0.02
hepatitis	2.1E+2	2.0E+2	3.0E+3 (4.0E+3)	0.02	6.7E+1	6.3E+1	3.0E+3 (4.0E+3)	0.06	0.08	0.08	0.23 (0.11)	0.00
hypothyroid	3.0E+4	2.9E+4	7.2E+5 (6.8E+4)	0.02	2.3E+4	2.3E+4	5.9E+4 (6.9E+4)	0.02	1.17	0.83	1.15 (0.40)	0.29
kr-vs-kp	2.9E+5	2.9E+5	3.7E+5 (9.2E+4)	0.00	1.6E+3	1.6E+3	2.4E+5 (1.4E+5)	0.00	1.14	1.12	2.28 (0.79)	0.01
lymph	1.4E+4	1.4E+4	1.5E+4 (7.9E+2)	0.00	5.8E+2	5.5E+2	8.7E+3 (4.0E+3)	0.05	0.47	0.48	0.43 (0.03)	−0.03
mushroom	9.5E+2	8.1E+2	1.1E+3 (1.7E+2)	0.14	2.7E+2	2.3E+2	9.2E+2 (2.3E+2)	0.15	0.23	0.23	0.57 (0.09)	0.00
primary-tumor	4.1E+3	4.1E+3	4.8E+3 (7.0E+2)	0.00	2.1E+2	2.1E+2	3.2E+3 (1.5E+3)	0.02	0.23	0.24	0.27 (0.03)	−0.01
soybean	5.9E+3	5.9E+3	7.8E+3 (1.1E+3)	0.00	4.6E+2	3.8E+2	5.2E+3 (2.0E+3)	0.17	0.27	0.28	0.31 (0.02)	−0.02
splice-1	2.9E+2	2.9E+2	5.9E+2 (2.5E+2)	0.00	4.4E+1	4.4E+1	5.7E+2 (2.5E+2)	0.00	0.65	0.66	0.67 (0.01)	−0.01
tic-tac-toe	5.6E+2	4.3E+2	2.0E+2 (1.5E+2)	0.23	5.5E+2	4.3E+2	2.0E+2 (1.5E+2)	0.23	0.10	0.11	0.06 (0.01)	−0.12
vote	9.5E+1	9.5E+1	9.3E+2 (6.6E+2)	0.00	4.6E+1	4.6E+1	9.2E+2 (6.6E+2)	0.00	0.05	0.06	0.12 (0.03)	−0.15
zoo-1	3.4E+2	3.3E+2	4.9E+2 (1.9E+2)	0.01	2.1E+2	2.0E+2	4.6E+2 (2.1E+2)	0.02	0.04	0.05	0.06 (0.02)	−0.09

Dataset	#Visited patterns (entire)				#Visited patterns (effective)				Running time (in seconds)			
	Static	Dynamic	Random	Ratio	Static	Dynamic	Random	Ratio	Static	Dynamic	Random	Ratio
anneal	9.0E+5	7.6E+5	7.5E+6 (9.1E+6)	0.16	8.9E+5	7.5E+5	7.1E+6 (9.1E+6)	0.15	2.69	2.93	45.76 (45.28)	−0.09
australian-credit	1.7E+5	1.4E+5	1.1E+7 (2.1E+6)	0.17	1.4E+4	6.6E+3	1.0E+7 (2.1E+7)	0.54	0.89	0.83	44.12 (68.77)	0.06
german-credit	2.3E+6	1.1E+6	3.2E+5 (1.9E+5)	0.51	2.3E+6	1.1E+6	3.2E+5 (1.9E+5)	0.51	20.16	5.15	6.42 (2.18)	0.74
heart-cleveland	3.2E+4	2.7E+4	4.5E+6 (4.7E+6)	0.16	1.8E+3	8.8E+2	4.5E+6 (4.7E+6)	0.50	0.70	0.70	17.39 (15.83)	0.01
hepatitis	3.1E+7	1.4E+7	7.7E+6 (6.2E+6)	0.54	3.1E+7	1.4E+7	7.7E+6 (6.2E+6)	0.54	117.56	42.75	20.52 (13.79)	0.64
kr-vs-kp	4.3E+5	4.3E+5	9.8E+5 (8.6E+5)	0.00	1.8E+3	1.7E+3	8.1E+5 (9.0E+5)	0.03	2.07	2.21	8.29 (8.41)	−0.06
lymph	2.1E+4	1.9E+4	4.4E+4 (1.5E+4)	0.06	3.3E+3	2.6E+3	3.8E+4 (1.6E+4)	0.21	0.51	0.52	1.01 (0.26)	−0.03
mushroom	2.0E+4	1.7E+4	1.0E+4 (3.0E+3)	0.16	2.0E+4	1.7E+4	1.0E+4 (3.0E+3)	0.16	1.02	0.93	1.40 (0.31)	0.09
primary-tumor	3.8E+4	2.4E+4	2.4E+4 (5.7E+3)	0.37	3.8E+4	2.4E+4	2.1E+4 (4.9E+3)	0.37	0.96	0.70	0.74 (0.20)	0.27
soybean	1.4E+4	1.4E+4	1.6E+4 (2.0E+3)	0.00	1.3E+4	1.3E+4	1.3E+4 (3.9E+3)	0.00	0.44	0.47	0.46 (0.04)	−0.05
splice-1	1.5E+3	1.5E+3	1.0E+4 (3.2E+3)	0.01	1.3E+3	1.3E+3	1.0E+4 (3.2E+3)	0.02	1.21	1.33	1.69 (0.29)	−0.10
tic-tac-toe	2.0E+3	1.4E+3	1.3E+3 (3.6E+2)	0.30	2.0E+3	1.4E+3	1.2E+3 (3.4E+2)	0.30	0.18	0.19	0.17 (0.02)	−0.06
vote	1.6E+5	8.0E+4	4.6E+4 (1.1E+4)	0.49	1.6E+5	7.9E+4	4.0E+4 (1.2E+4)	0.50	1.61	1.45	0.88 (0.16)	0.10
zoo-1	2.7E+3	2.6E+3	2.1E+3 (5.3E+2)	0.01	2.2E+3	2.2E+3	1.9E+3 (5.4E+2)	0.02	0.17	0.19	0.18 (0.03)	−0.09

[2] G. Dong and J. Bailey, Eds., *Contrast Data Mining: Concepts, Algorithms, and Applications*. CRC Press, 2012.

[3] P. Kralj Novak, N. Lavrač, and G. I. Webb, "Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining," *J. of Machine Learning Research*, vol. 10, pp. 377–403, 2009.

[4] J. Han, J. Wang, Y. Lu, and P. Tzvetkov, "Mining top-k frequent closed patterns without minimum support," in *Proc. of ICDM-02*, 2002, pp. 211–218.

[5] S. Morishita and J. Sese, "Traversing itemset lattices with statistical metric pruning," in *Proc. of PODS-00*, 2000, pp. 226–236.

[6] R. Bayardo, R. Agrawal, and D. Gunopulos, "Constraint-based rule mining in large, dense databases," *Data Mining and Knowledge Discovery*, vol. 4, pp. 217–240, 2000.

[7] W. Li, J. Han, and J. Pei, "CMAR: Accurate and efficient classification based on multiple class-association rules," in *Proc. of ICDM-01*, 2001, pp. 369–376.

[8] G. I. Webb, "Discovering significant patterns," *Machine Learning*, vol. 68, pp. 1–33, 2007.

[9] Y. Kameya and T. Sato, "RP-growth: Top-k mining of relevant patterns with minimum support raising," in *Proc. of SDM-12*, 2012, pp. 816–827.

[10] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. of SIGMOD-00*, 2000, pp. 1–12.

[11] C. C. Aggarwal, *Data Mining: The Textbook*. Springer, 2015.

[12] J. Li, H. Li, L. Wong, J. Pei, and G. Dong, "Minimum description length principle: generators are preferable to closed patterns," in *Proc. of AAAI-06*, 2006, pp. 409–414.

[13] Y. Kameya and H. Asaoka, "Depth-first traversal over a mirrored space for non-redundant discriminative itemsets," in *Proc. of DaWaK-13*, 2013, pp. 196–208.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[15] G. I. Webb, "OPUS: An effficient admissible algorithm for unordered search," *J. of Artificial Intelligence Research*, vol. 3, pp. 431–465, 1995.

[16] M. Atzmueller and F. Lemmerich, "Fast subgroup discovery for continuous target concepts," in *Proc. of ISMIS-09*, 2009, pp. 35–44.

[17] R. Srikant and R. Agrawal, "Mining generalized association rules," in *Proc. of VLDB-95*, 1995, pp. 407–419.