Margin-Closed Frequent Sequential Pattern Mining

Dmitriy Fradkin
Siemens Corporate Research
755 College Road East
Princeton, NJ 08540
dmitriy.fradkin@siemens.com

Fabian Moerchen
Siemens Corporate Research
755 College Road East
Princeton, NJ 08540
fabian.moerchen@siemens.com

ABSTRACT

We present a new approach to mining sequential patterns that significantly reduces the number of patterns reported, favoring longer patterns and suppressing shorter patterns with similar frequencies. This is achieved by mining only margin-closed patterns whose support differs by more than some margin from any extension. Our approach extends the efficient BIDE algorithm to enforce the margin constraint. The set of margin-closed patterns can be significantly smaller than a set of just closed patterns while retaining the most important information about the dataset. This is shown by an extensive empirical evaluation on six real life databases.

1. INTRODUCTION

Temporal data mining exploits temporal information in data sources in the context of data mining tasks such as clustering or classification. Many scientific and business data sources are dynamic and thus promising candidates for application of temporal mining methods. For an overview of methods to mine time series, sequence, and streaming data see [17, 14].

One particular type of temporal data are sequences of (sets of) discrete items associated with time stamps, for example histories of transactions of customers in an online shop or log messages emitted by machines or telecommunication equipment during operation. A common task is to mine for local regularities in this data by looking for sequential patterns [2] that represent a sequence of itemsets possibly with gaps in the observation sequences.

It is well know that frequent itemset mining suffers from a combinatorial explosion of results when lowering the minimum support threshold. When mining sequential patterns, i.e., sequences of itemsets on sequential databases, this effect typically becomes even stronger. A lossless way of reducing the number of reported patterns that favors longer, thus more interpretable patterns, is mining of closed patterns. Only patterns that cannot be extended with additional items without lowering their support are reported. A straightforward extension of closed itemset mining are margin-closed itemsets, also known as δ-tolerance itemsets [12]. A margin closed pattern cannot be extended by additional items without lowering the support significantly, as determined by a relative or absolute threshold.

In this work we present an efficient algorithm for mining of margin-closed sequential patterns. The well known BIDE (BI-Directional Extension checking) [38] algorithm is extended to enforce the margin-closed constraints. We show that on real life data the number of reported patterns can be greatly reduced even with moderate margin thresholds. Using a classifier we show that the suppressed (almost redundant) patterns were not of great importance for a specific data mining task.

Related work, mostly in the area of itemset and sequence mining, is described in Section 2. The technical part describes preliminary definitions (Section 3.1), the original BIDE algorithm (Section 3.2) and the proposed extension BIDE-Margin (Section 3.3). Section 4 contains results of evaluation on real life data. Conclusion is presented in Section 5.

2. RELATED WORK

Many publications explored the question of reducing the number of patterns within a general pattern mining framework [20, 6]. In the sections below we discuss methods focused on itemset and sequential mining as being most relevant to our work.

2.1 Itemset mining

Researchers have proposed many solutions to reduce the number of itemset patterns depending on the context in which the patterns are used, for example, condensed representations [8], constrained itemsets [32] and combinations thereof [4, 13], and compression [37, 36]. For association rule generation, closed itemsets [31, 5] are commonly used to avoid redundant rules [42] favoring longer patterns to generate specific rules. For frequency queries non-derivable itemsets [7] provide a compact lossless representation favoring shorter patterns to keep the summary small.

Margin-closed itemsets have been previously proposed by the authors for exploratory knowledge discovery tasks in the context of temporal data mining [25, 27] and independently as δ-tolerance itemsets for frequency estimation in [12]. Margin-closed patterns are a specialization of closed itemsets with a constraint to limit the redundancy among reported patterns. An itemset is closed if no superset with the same frequency exists. An itemset is margin-closed if no superset with almost the same frequency exists, where 'almost' is defined by a threshold α on the relative (or absolute) difference of the frequencies. The threshold ensures a frequency *margin* among the reported patterns. An efficient algorithm for mining margin-closed itemsets, extending the well-know DCI_Closed algorithm [21], has been proposed in [24].

A related line of work is motivated by the fact that transaction data is often noisy. The strict definition of support, requiring all

UP'10, July 25th, 2010, Washington, DC, USA.

items of an itemset to be present in a transaction, is relaxed, see [16] and references therein. These approaches can reveal important structures in noisy data that might otherwise get lost in a huge amount of fragmented patterns. One needs to be aware though that such approaches report approximate support values and possibly list itemsets that are not observed as such in the collection at all [1] or with much smaller support.

2.2 Sequential mining

An overview of algorithms for sequential pattern mining is given in [43]. Our approach extends the BIDE algorithm [38] that uses a smart search space pruning and does not require the patterns found to remain in memory until the algorithm terminates.

Motivated by approaches that have worked on itemsets, research on reduction of the output of sequential pattern mining algorithms includes compression of the mining result in a post-processing step [10, 40], a condensed representation to evaluate sequential association rules [34], and approximate patterns [44] under the Hamming distance.

These approaches are different from the one presented here in at least one of the following ways:

- The margin constraint favors longer patterns, whereas condensed representation focus at reconstruction of frequencies for patterns not reported or compression ratio of the complete pattern set.
- Patterns observed exactly as is with exact frequencies are reported, whereas approximate patterns represent observations that may differ (slightly).
- The pruning is integrated in the mining algorithm whereas compression is a postprocessing of the results after mining.

The presented approach therefore has merits in particular if the patterns are used in a context that requires interpretation, as opposed to automated post processing with other algorithms. Longer patterns are more interpretable because they offer more context to the analyst. While approximations that tolerate errors may be more robust, they may report approximate support values and possibly list itemsets that are not observed as such in the collection at all or with much smaller support. This might be misleading in exploratory applications.

A generalization of sequential patterns are partial orders [9]. Instead of requiring a full ordering of the itemsets in a pattern some order relation may be unspecified. This is typically represented by a directed graph of itemsets. In [33] it is shown that closed partial orders are also a generalization of Episodes [22] that are restricted to combinations of fully ordered and completely unordered patterns.

In [9] closed partial orders are mined by grouping of sequential patterns and generating directed graphs. In [26] it is shown that the grouping corresponds to an instance of the closed itemset mining problem. [35] describes an algorithm to mine arbitrary (not necessarily closed) groups of sequential patterns. Experiments on real life data in [29] show that the grouping can both reduce or increase the number of patterns found, depending on the dataset.

3. MINING MARGIN-CLOSED SEQUENTIAL PATTERNS

3.1 Preliminaries

DEFINITION 3.1. *An event sequence over a set of events Σ is a sequence of pairs (t_i, s_i) of event sets $s_i \subseteq \Sigma\ \forall i = 1, \ldots, n$ and time stamps $t_i \in \Re^+$. The ordering is based on time, i.e. $\forall i < j : t_i \leq t_j$. The length of the event sequence is n.*

For most of the discussion in our work (and in much of sequential pattern mining literature) the exact values of the time stamps are not as important as the ordering that they impose. We will therefore omit timestamps from discussion and will treat event sequences as just an ordered set of event sets $S = \{s_i\}$.

DEFINITION 3.2. *A sequence database, SDB, of size N is a collection of event sequences P_i, $i = 1, \ldots, N$.*

We now need to introduce a basic definition from order theory.

DEFINITION 3.3. *A partial order is a binary relation $\prec$ over a set S which is reflexive, antisymmetric, and transitive, i.e., for all $a, b, c \in P$, we have that:*

- *$a \prec a$ (reflexivity);*
- *$a \prec b$ and $b \prec a$ imply $a = b$ (antisymmetry);*
- *$a \prec b$ and $b \prec c$ imply $a \prec c$ (transitivity).*

A set S with a partial order is a chain iff $\forall a, b \in S$: $a \prec b$ or $b \prec a$.

DEFINITION 3.4. *A partial order pattern P is a set of event sets $\{p_i\}$,$i = 1, \ldots, n$ together with a partial order $\prec$ over them.*

DEFINITION 3.5. *A sequential pattern P is a partial order pattern that is a chain: $p_1 \prec p_2 \prec \ldots \prec p_n$.*

Note that in Episode mining sequential patterns are called serial patterns.

DEFINITION 3.6. *A parallel pattern P is a partial order pattern with no order relations among the event sets.*

DEFINITION 3.7. *A sequence $S = \{s_i\}, i = 1, ..., k$ matches a sequential pattern $P = \{p_j\}, i = 1, ..., m$ (or a pattern occurs in the sequence) iff $\exists i_1, ..., i_m$ with $p_j \subseteq s_{i_j}$ for $j = 1, ..., m$, such that $\forall 1 \leq j, k \leq m$: $p_j \prec p_k$ implies $i_j < i_k$. We will denote such a match by $m_{i_1, i_m}(P, S)$.*

DEFINITION 3.8. *A match $m_{i_1,i_m}(P, S)$ is the earliest match iff for any other match $m_{j_1,j_m}(P, S)$ $i_k \leq j_k$, $\forall k = 1, \ldots, m$,*

DEFINITION 3.9. *A pattern P has support$(P) = s$ in an SDB D if D contains s distinct event sequences that match P. A pattern is frequent iff its its support is no less than some predefined minimum support value μ, i.e. $support(P) >= \mu$.*

In the following, when talking about patterns, we will always assume that they are frequent, with some minimum support μ defined.

DEFINITION 3.10. *A frequent pattern P is considered closed in an SDB D, if is there is no pattern $P' \neq P$ in D, such that $\exists m(P, P')$ (i.e. P occurs in P') and support(P') = support(P).*

DEFINITION 3.11. *A pattern P is considered margin-closed in an SDB D, with margin α, if is there is no pattern $P' \neq P$ in D such that P occurs in P' and support$(P') > (1-\alpha) *$ support(P).*

In other words, P is margin-closed if there is no pattern P' that contains P and is almost as frequent.

In order to describe the algorithms in the following sections, we need to introduce the notion of *projected databases*, which is extremely useful in constructing efficient algorithms for sequential pattern mining.

DEFINITION 3.12. *Given a pattern $P = \{p_i\}$, $i = 1, \ldots, |P|$ and a sequence $S = \{s_j\}$,$j = 1, \ldots, |S|$, with the earliest match $m_{k_1,k_m}(P, S)$, a projection of S on P results in a projected sequence $S|P = \{s_t\}$, where $t = k_m + 1, \ldots, |S|$. We refer to k_m as an offset.*

DEFINITION 3.13. *Given a pattern $P = \{p_i\}$, $i = 1, \ldots, |P|$ and an SDB $D = \{S_j\}$,$j = 1, \ldots, |D|$, a projection of D on P is a projected database $D|P$, consisting of projected sequences $S_{j_u}|P$, obtained by projecting S_{j_u} onto P. Note that if a sequence does not match a pattern, it does not appear in the projected database.*

Projected database $D|P$ can be efficiently represented with a list of pairs of indices (j_u, t_u), where j_u refers to $S_{j_u}|P$ and t_u is the corresponding offset.

DEFINITION 3.14. *For a projected sequence $S|P$ we define two operations:*

- *$original(S|P) = S$; and*
- *$prefix(S|P) = \{s_t\}$, $t = 1, \ldots, k_m$, where $m_{k_1,k_m}(P, S)$ is the earliest match.*

in other words, $original$ of a projection returns the whole sequence S, while $prefix$ of a projection returns the part of the sequence preceeding the projection.

3.2 The BIDE algorithm

BIDE is an efficient algorithm for finding frequent closed sequential patterns in sequential databases [38]. We extend this algorithm to enforce the margin-closed constraints. In order to make this paper self-contained we provide a detailed description of BIDE using our own definitions.

BIDE is initially called with the full sequential database D, minimum support μ and an empty pattern $P = \emptyset$. It returns a list of frequent closed sequential patterns. BIDE operates by recursively extending patterns, and, while their frequency is above the minimum support, checking closure properties of the extensions.

Consider a frequent pattern $P = \{p_i\}$,$i = 1, \ldots, n$. There are two ways to extend pattern P *forward* with item j:

- Appending the set $p_{n+1} = \{j\}$ to P obtaining $P' = p_1 \ldots p_n p_{n+1}$, called a forward-S(equence)-extension.
- Adding j to the last itemset of P: $P' = p_1 \ldots p'_n$, with $p'_n = p_n \cup j$, assuming $j \notin p_n$, called a forward-I(tem)-extension.

Similarly, a pattern can be extended *backward*

- Inserting the set $p_x = \{j\}$ into P anywhere before the last set obtaining $P' = p_1 \ldots p_i p_x p_{i+1} \ldots p_n$, for some $0 \leq i \leq n$, called a backward-S(et)-extension.
- Adding j to any set in P obtaining $P' = p_1 \ldots p'_i \ldots p_n$, with $p'_i = p_i \cup j$, assuming $j \notin p_n$, $1 \leq i \leq n$, called a backward-I(tem)-extension.

According to a Theorem 3 of [39], a pattern is closed if there exists no forward-S-extension item, forward-I-extension item, backward-S-extension item, nor backward-I-extension item with the same support.

Furthermore, if there is a backwards extension item, then the resulting extension and all of its future extension are explored in a different branch of recursion, meaning that it can be pruned from current analysis. These insights are combined in BIDE, leading to

Algorithm 1 BIDE Algorithm

Require: Sequential Pattern $P = \{p_i\}$, Projected Database $D|P$, minimum support μ

F - set of frequent closed patterns
$l = |P|$
$Ls = sStepFrequentItems(P, D|P, \mu)$;
$Li = iStepFrequentItems(P, D|P, \mu)$;
if !($frequencyCheck(Ls, P)$ || $frequencyCheck(Li, P)$) **then**
    if $backscan(P', D', true)$ **then**
        $F = F \cup P$
    end if
end if
for itemset $p \in Ls$ **do**
    $P' = p_1, .., p_l, p$
    if $backscan(P', D|P', false)$ **then**
        $F = F \cup bide(P', D', \mu)$;
    end if
end for
for itemset $p \in Li$ **do**
    $P' = p_1, .., p_{l-1}, p_l \cup p$
    if $backscan(P', D|P', false)$ **then**
        $F = F \cup bide(P', D', \mu)$;
    end if
end for
return F

Algorithm 2 FrequencyCheck

Require: Pattern P, HashMap M of forward (I or S) extension items with their supports

for $i \in M$ **do**
    if M(i)=support(P) **then**
        return true
    end if
end for
return false

Algorithm 3 Finding forward-S-expansion Candidates

Require: Sequential Pattern $P = \{p_i\}$; Projected Database $D|P$, minimum support μ

Initialize Hash Map M
for $i = 1, \ldots, |D|P|$ **do**
    $I = \emptyset$
    for $j = 1, \ldots, |s_i|$ **do**
        $I = I \cup s_{ij}$
    end for
    for item $\in I$ **do**
        $M(i) = M(i) + 1$
    end for
end for
for $i \in M$ **do**
    if $M(i) < \mu$ **then**
        Delete $M(i)$
    end if
end for
return M

Algorithm 4 Finding forward-I-expansion Candidates

```
Require: Sequential Pattern P = {p_i}; Projected Database D|P, μ
Initialize Hash Map M
Let l = |P|
for i = 1, ..., |D|P| do
    I = ∅
    for j = 1, ..., |s_i| do
        if p_l ∈ s_i then
            I = I ∪ s_ij
        end if
    end for
    for item i ∈ I do
        M(i) = M(i) + 1
    end for
end for
for i ∈ M do
    if M(i) < μ then
        Delete M(i)
    end if
end for
return M
```

a very memory-efficient algorithm, because the patterns found do not need to be kept in memory while the algorithm is running.

Specifically, consider pseudo-code for BIDE (Algorithm 1). In Lines 3-4 items that can be used in forward extension of the current pattern are found. If there is no forward extension with the same support (Line 5), the backward closure is checked (Line 6) using function backScan. If the pattern is also backwards-closed, it can be added to the set of closed frequent patterns (Line 7).

Then, we check every item in forward S and I extensions (in the two for-loops) to see whether it is explored in a different branch of recursion, again via backScan function (Lines 12 and 18). If not, then we project the database on the extension and call BIDE recursively on the extension and the new projected database.

Pseudo-code for $sStepFrequentItems$ and $iStepFrequentItems$ is shown in Algorithms 3 and 4. Algorithm 2 shows the FrequencyCheck function. These functions are rather straightforward.

Algorithm 5 BackScan Function. If $closedCheck$ is true, checks if P is closed. If $closedCheck$ is false, checks if P is examined in a different branch of the recursion.

```
Require: Sequential Pattern P = {p_i}, Projected Database D|P, μ, closedCheck
Initialize a 3D integer array G (for gaps)
for i = 1, ..., |D| do
    if closedCheck then
        G[i] = FindMaximumGaps(P, original(S_i))
    else
        G[i] = FindMaximumGaps(P, prefix(S_i))
    end if
end for
if BackwardIExpansionCheck(P, D|P, μ, closedCheck, G) then
    if BackwardSExpansionCheck(P, D|P, μ, closedCheck, G) then
        return true
    end if
end if
return false
```

Algorithm 6 FindMaximumGaps

```
Require: Sequential Pattern P = {p_i}; Event Sequence S
Initialize |P| × 2 integer array G
if ∃m_{i_1,i_m}(P, S) then
    G[0][0] = 0
    for j = 1, ..., |P| do
        Set G[j][0] = i_j
    end for
    Compute S^r - reverse of S
    Compute P^r - reverse of P
    Compute m_{i_1,i_m}(P^r, S^r)
    for j = 1, ..., |P^r| do
        Set G[|P| - j][1] = |S| - i_j
    end for
end if
return G
```

It remains to discuss the $backScan$ function (Algorithm 5). The backScan function has two uses. The first time it is called in function BIDE, closure check flag is set to true (Line 6). Then backScan returns true if and only if pattern P is backwards closed, i.e. if there is no backwards extension with the same support. This is **Case I**. The other calls from BIDE are with closure check set to false. In these situations backScan needs to check if a pattern extension is backwards closed in its projected database i.e. that it can't be reached in a different way, via a different recursion branch. This is **Case II**.

In order to check for backward extensions of a pattern P, we need to know which parts of sequences in D need to be looked at. If P has a backward-S-extension item t between p_i and $p_i + 1$, it means that in each sequence $S \in D$ matching P there is a particular match $m_{k_1,k_m}(P, S)$, such that t occurs between s_{k_i} and $s_{k_{i+1}}$. In order to check for an existence of such an item, we can find the earliest and the latest matches $m_{k_1,k_m}(P, S)$ and $m_{j_1,j_m}(P, S)$, and examine the itemsets $s_{k_i+1}, \ldots, s_{j_{i+1}-1}$. In other words, we check the itemsets between the earliest occurrence in a match of p_i and the latest occurrence in a match of p_{i+1}. Similarly, we can check for existence of a backward-I-extension item t by looking at all possible occurrences of t together with p_i, starting from earliest and ending with latest match occurrences of p_i. Function $FindMaximumGaps$ (Algorithm 6) is used exactly for finding and storing the earliest and latest indices of consecutive itemsets of pattern P in a match $m(P, S)$. Finding of the latest match is most efficiently found by searching for a reverse of P in a reverse of sequence S, and transforming the indices appropriately.

We can now discuss the two Cases mentioned above. In **Case I** closure check flag is set to true. Since we want to check if pattern P is closed, we need to fully examine all sequences in $D|P$ for potential extensions. Therefore, function FindMaximumGaps is called on original sequences in $D|P$, not on the projections. In **Case II**, we only care if there is a backward extension in order to prune the current pattern. Therefore, when $closedCheck$ is false, FindMaximumGaps is called on prefixes of sequences in $D|P$.

Once array G is computed, we check for I-expansions and for S-expansions (Algorithms 7 and 8). Consider BackwardIExpansionCheck. We want to detect if an item can be inserted into any itemset of P, while maintaining the same support. Therefore, for each position in P, we examine all sequences in $D|P$, in the intervals specified by array G. If $closedCheck$ is true, we look at the full sequence S, otherwise we look at the prefix of $S|P$. The 'end' indices need to be computed differently for the two cases, because when $closedCheck$ is false, the last occurrence of the last itemset of P is also the first potential point for forward expansion and does not need to be considered, but when $closedCheck$ is true, we check for closedness of P and all potential expansion locations need to be examined.

For each s_j, if $p_i \in s_j$, we add all items in s_j to set C, except for items already in p_i. After processing a sequence, we update frequency counts of items in C that are stored a hash map M, and keep track of the maximum frequency value. Once we have processed all sequences for a particular p_i, we check if the maximum frequency is equal to support of P ($support(P) = |D|P|$). If so, that means that there is some backward-I-expansion item for P, and therefore P is not closed and, there is another recursion branch that will examine this expansion, so BackwardIExpansionCheck returns false. If maximum frequency is below support of P, BackwardIExpansionCheck return true.

BackwardSExpansionCheck operates similarly.

Algorithm 7 BackwardIExpansionCheck

```
Require: Pattern P, Projected Database D|P, μ, closedCheck,
    gap array G
for i = 1, ..., |P| do
    Initialize Hash Map M
    for j = 1, ..., |D| do
        start = G[j][i][0] + 1
        if closedCheck then
            end = G[j][i][1]
            S' = original(S_j)
        else
            end = G[j][i][1] − 1
            S' = prefix(S_j)
        end if
        C = ∅
        for k = start, ..., end do
            if P_i ∈ S'_k then
                C = C ∪ S'_k
            end if
        end for
        C = C − P_i
        max = 0
        for s_i ∈ C do
            M(s_i) = M(s_i) + 1
            if M(s_i) > max then
                max = M(s_i)
            end if
        end for
    end for
    if max = support(P) then
        return false;
    end if
end for
return true
```

3.3 The BIDE-Margin algorithm

We now describe the changes required to enforce margin-closedness in BIDE leading to the BIDE-Margin algorithm. The flag $marginCheck$ is used in the backScan function instead of $closedCheck$ and there is the additional margin parameter α. There are three changes to the functions described in the following sections.

When Forward Expansion is considered, rather than checking if there are items with the same support as the current pattern, one instead checks for presence of items that are within margin α of the pattern's support. The function FrequencyCheck for BIDE-Margin is shown in Algorithm 9.

Algorithm 8 BackwardSExpansionCheck

```
Require: Pattern P, Projected Database D|P, μ, closedCheck,
    gap array G
for i = 1, ..., |P| do
    Initialize Hash Map M
    for j = 1, ..., |D| do
        start = G[j][i][0] + 1
        end = G[j][i][1] − 1
        if closedCheck then
            S' = original(S_j)
        else
            S' = prefix(S_j)
        end if
        C = ∅
        for k = start, ..., end do
            C = C ∪ S'_k
        end for
        max = 0
        for s_i ∈ C do
            M(s_i) = M(s_i) + 1
            if M(s_i) > max then
                max = M(s_i)
            end if
        end for
    end for
    if max = support(P) then
        return false;
    end if
end for
return true
```

Algorithm 9 FrequencyCheck for BIDE-Margin

```
Require: Pattern P, HashMap M of forward (I or S) extension
    items with their supports, α
for i ∈ L do
    if M(i) >= (1 − α) * support(P) then
        return true
    end if
end for
return false
```

The other two changes involve checking backward closure. We need to check if there are any items that are margin-close to the pattern, and if so then the pattern is not margin-closed. This leads to changes to BackwardIExpansionCheck and BackwardSExpansionCheck functions. Algorithms 10 and 11 respectively show how these algorithms need to be modified for BIDE-Margin. The parameter $marginCheck$ replaces $closedCheck$, and is used similarly, except for additions in Lines 25-29 and Lines 21-25. When $marginCheck$ is true we check if pattern is margin-closed, and therefore if there is an item with frequency above μ and within margin of the support of P, we know that P is not margin-closed. Note that when $marginCheck$ is false, this check should not be performed - we cannot disregard the recursion branches going from the current pattern unless there is a backward extension with exactly the same support.

3.4 Computational Efficiency

The BIDE-Margin algorithm has the same complexity as BIDE,

Algorithm 10 BackwardIExpansionCheck for BIDE-Margin

```
Require: P, D|P, μ, marginCheck, G, α
for i = 1, ..., |P| do
    Initialize Hash Map M
    for j = 1, ..., |D| do
        start = G[j][i][0] + 1
        if marginCheck then
            end = G[j][i][1]
            S' = original(S_j)
        else
            end = G[j][i][1] − 1
            S' = prefix(S_j)
        end if
        C = ∅
        for k = start, ..., end do
            if P_i ∈ S'_k then
                C = C ∪ S'_k
            end if
        end for
        C = C − P_i
        max = 0
        for s_i ∈ C do
            M(s_i) = M(s_i) + 1
            if M(s_i) > max then
                max = M(s_i)
            end if
            if marginCheck then
                if max ≥ (1 − α) ∗ |D|P| AND max ≥ μ then
                    return false
                end if
            end if
        end for
    end for
    if max = |D| then
        return false;
    end if
end for
return true
```

Algorithm 11 BackwardSExpansionCheck for BIDE-Margin

```
Require: P, D|P, μ, marginCheck, G, α
for i = 1, ..., |P| do
    Initialize Hash Map M
    for j = 1, ..., |D| do
        start = G[j][i][0] + 1
        end = G[j][i][1] − 1
        if marginCheck then
            S' = original(S_j)
        else
            S' = prefix(S_j)
        end if
        C = ∅
        for k = start, ..., end do
            C = C ∪ S'_k
        end for
        max = 0
        for s_i ∈ C do
            M(s_i) = M(s_i) + 1
            if M(s_i) > max then
                max = M(s_i)
            end if
            if marginCheck then
                if max ≥ (1 − α) ∗ |D|P| AND max ≥ μ then
                    return false
                end if
            end if
        end for
    end for
    if max = |D| then
        return false;
    end if
end for
return true
```

since it still generates all frequent sequential patterns, in exactly the same fashion. However, unlike BIDE it searches for margin-closed, rather than just closed, patterns and therefore it will need to check closeness less frequently. In other words, due to a "looser" frequency check used by BIDE-Margin (Algorithm 9), the call to $backscan$ algorithm in Line 6 of Algorithm 1 will occur less frequently in BIDE-Margin than in BIDE. Thus, while the overall algorithm complexity is the same, BIDE-Margin may perform slightly faster. The extent of this depends on the nature of the data and the value of margin specified.

We would also like to note that BIDE-Margin is significantly more efficient than a brute force post-processing of BIDE results would be. If N is the number of patterns produced by BIDE for a particular value of support, the postprocessing would require $O(N)$ memory and $O(N^2)$ time to order the patterns by their support and then to check for each pattern P if it is margin-closed.

4. EXPERIMENTS

We performed experiments on real life sequential data sets to compare BIDE and BIDE-Margin in two ways: 1) The number of patterns produced. By definition BIDE-Margin with $\alpha > 0$ produces the same or less patterns than BIDE. The goal of the experiment is to quantify the extent of this reduction on real life data. 2) Predictiveness of patterns found: We compare the classification performance of the sets of patterns when used as features in SVM training. Since BIDE-Margin suppresses only features that are very similar in frequency to reported features, we expect to see only minor performance decrease, if any. With SVM being a classifier that can deal with high dimensional data and redundancy among the features this is a tough test.

We did not perform run-time comparisons between BIDE and BIDE-Margin, since the differences are expected to be small. Similarly, the scalability with the number of sequences in the database is inherited directly from BIDE.

4.1 Data

We performed experiments on six interval datasets, previously used in [29], summarized in Table 1. While technically databases of intervals, they can be interpreted as sequential databases by treating start and end boundaries of an interval as separate events [41]. Specifically, each symbolic interval, a triple (t_s, t_e, σ) with event $\sigma \in \Sigma$ and time stamps $t_s \leq t_e$, is converted into two symbolic time points (t_s, σ^+) and (t_e, σ^-), and then all time points with the same time stamp are aggregated into itemsets, resulting in a standard event sequence as in Definition 3.1. Further details are given in [29].

The advantage of this collection is that class labels are available for each sequence that allows an automated evaluation of patterns using a classifier, while the categorical sequential data available in the UCI Machine Learning Repository [3] is largely unlabeled such as web log data.

Data	Intervals	Labels	Sequences	Classes
ASL-BU	18250	154	441	7
Auslan2	900	12	200	10
Blocks	1207	8	210	8
Context	12916	54	240	5
Pioneer	4883	92	160	3
Skating	18953	41	530	6/7

Table 1: Interval data: Seven databases consisting of many sequences of labeled intervals with class labels for each sequence.

ASL-BU[1] The intervals are transcriptions from videos of American Sign Language expressions provided by Boston University [30]. It consists of observation interval sequences with labels such as *head mvmt: nod rapid* or *shoulders forward* that belong to one of 7 classes like *yes-no question* or *rhetorical question*.

Auslan2 The intervals were derived from the high quality Australian Sign Language dataset in the UCI repository [3] donated by Kadous [19]. The x,y,z dimensions were discretized using Persist with 2 bins, 5 dimensions representing the fingers were discretized into 2 bins using the median as the divider. Each sequence represents a word like *girl* or *right*.

Blocks[2] The intervals describe visual primitives obtained from videos of a human hand stacking colored blocks provided by [15]. The interval labels describe which blocks touch and the actions of the hand (*contacts blue red*, *attached hand red*). Each sequence represents one of 8 different scenarios from atomic actions (*pick-up*) to complete scenarios (*assemble*).

Context[3] The intervals were derived from categoric and numeric data describing the context of a mobile device carried by humans in different situations [23]. Numeric sensors were discretized using 2-3 bins chosen manually based on exploratory data analysis. Each sequence represents one of five scenarios such as *street* or *meeting*.

Pioneer The intervals were derived from the Pioneer-1 datasets in the UCI repository [3]. The numerical time series were discretized into 2-4 bins by choosing thresholds manually based on exploratory data analysis. Each sequence describes one of three scenarios: *gripper*, *move*, *turn*.

Skating The intervals were derived from 14 dimensional numerical time series describing muscle activity and leg position of 6 professional In-Line Speed Skaters during controlled tests at 7 different speeds on a treadmill [27]. The time series were discretized into 2-3 bins using Persist and manually chosen thresholds. Each sequence represents a complete movement cycle and is labeled by skater or speed.

4.2 Numerosity

By construction, the number of patterns produced by BIDE-Margin is always less than or equal to that produced by BIDE. Figure 1 shows the number of patterns (on a log10 scale) found by these methods using different support thresholds and margin values. For all datasets except ASL-BU, BIDE-Margin produces significantly fewer patterns. The reduction is strongest for the Context and Skating data and for Auslan2 for large minimum supports.

4.3 Predictiveness

Patterns obtained by unsupervised mining can be used for knowledge discovery by ranking and analyzing them directly, for generation of temporal association rules [18], or as features in predictive models [11]. We analyzed the predictiveness of the patterns by estimating classifier performance with each set of patterns.

In our experiments we have used the Spider Toolbox for Matlab[4] As classifier, we focused on Support Vector Machines. We have also experimented with decision trees and random forests, obtaining qualitatively similar results.

Once patterns were generated, for a particular value of support and margin, we have performed 10-fold cross-validation with linear SVM, setting parameter C in turn to 2^k, $k = -10, -9, \ldots, 9, 10$. The best value over all values of C is reported. Note, that this is done purely for the purpose of comparing the properties of two unsupervised pattern mining techniques, hence we did not interleave the pattern mining with the cross validation, as would be needed if the goal were to train a classifier with good generalization performance.

The results are shown in Figure 2. The y-axis is the lowest classification error, while the x-axis is the minimum support. The results with different margin values are shown as different lines. Examination of these results suggests that using margin 0.05 or 0.1 barely affects the classification error rate. Margin of 0.2 does lead to noticeably worse results on Pioneer dataset, and on Auslan2 with support 20, but not on the other datasets. The differences in performance tend to become smaller as support increases and the number of patterns decreases.

Figure 3 shows results obtained with J48, using the default settings. The results are qualitatively similar to those obtained with SVM, i.e. classification error does not increase for small values of the margin. Results with random forests are omitted due to space constraints.

5. CONCLUSION

We presented a new constraint for reducing the output of sequential pattern mining and an efficient algorithm for mining such patterns. We have demonstrated that the number of margin-closed patterns can be a lot smaller than that of closed patterns, but that these patterns are just as useful, as evidenced by performance of classifiers built using these patterns.

Using data mining in real life systems often requires the analyst to understand and trust the reported results to take appropriate action. We believe that reporting of exact patterns with exact frequency and favoring longer patterns while pruning similar shorter patterns are all advantageous for interpretation of mining results by an analyst. For some domains, however, error tolerance [44] may be more important than interpretability.

Mining closed sequential patterns is an important task in temporal data mining from time point and time interval data [28] and it is a substep in the process of mining partial orders [9]. In future work, we intend to conduct an experimental evaluation of the run-time of BIDE-Margin compared with BIDE using both actual pattern mining time and the time needed by follow up data mining tasks such as classification or grouping of sequences into partial orders.

6. REFERENCES

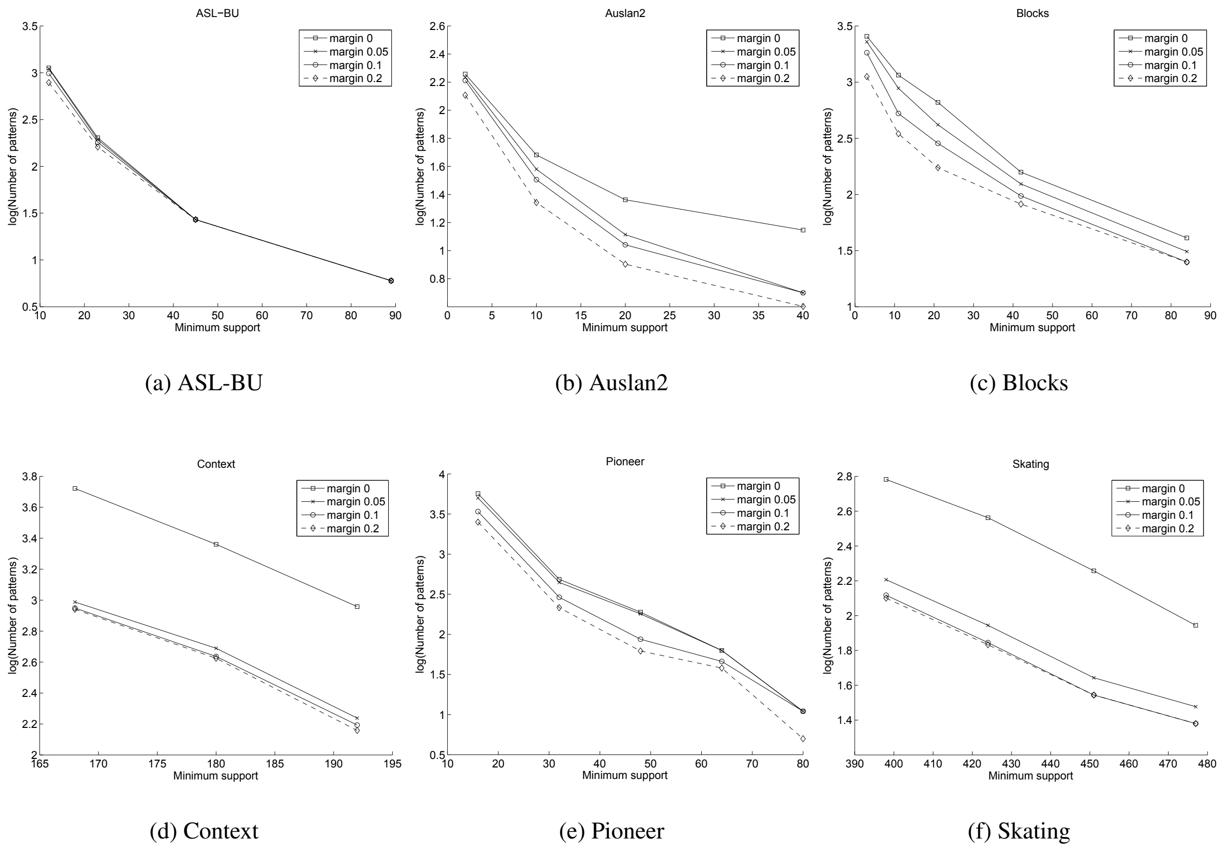
[1] F. Afrati, A. Gionis, and H. Mannila. Approximating a collection of frequent sets. In *Proc. 10th ACM SIGKDD Int.*

[1] http://www.bu.edu/asllrp/

[2] ftp://ftp.ecn.purdue.edu/qobi/ama.tar.Z

[3] http://www.cis.hut.fi/jhimberg/contextdata/index.shtml

[4] http://www.kyb.mpg.de/bs/people/spider/main.html



(a) ASL-BU (b) Auslan2 (c) Blocks

(d) Context (e) Pioneer (f) Skating

Figure 1: Comparison of the number of patterns (log10 scale) for different minimum support thresholds and margin values.

Conf. on Knowledge Discovery and Data Mining, pages 12–19. ACM Press, 2004.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proc. IEEE ICDE*, pages 3–14. IEEE Press, 1995.

[3] A. Asuncion and D.J. Newman. UCI Machine Learning Repository. University of California, Irvine `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[4] F. Bonchi and C. Lucchese. On condensed representations of constrained frequent patterns. *Knowl. Inf. Syst.*, 9(2):180–201, 2006.

[5] J.-F. Boulicaut and A. Bykowski. Frequent closures as a concise representation for binary data mining. In *Proc. Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 62–73, 2000.

[6] Björn Bringmann and Albrecht Zimmermann. One in a million: picking the right patterns. *Knowledge and Information Systems*, 18(1):61–81, 2008.

[7] T. Calders and B. Goethals. Non-derivable itemset mining. *Data Mining and Knowledge Discovery*, 14(1):171–206, 2007.

[8] T. Calders, C. Rigotti, and J.-F. Boulicaut. A survey on condensed representations for frequent sets. In *Constraint-Based Mining and Inductive Databases*, pages 64–80, 2006.

[9] G. Casas-Garriga. Summarizing sequential data with closed partial orders. In *Proc. of the 5th SIAM Intl. Conf. on Data Mining (SDM)*, pages 380–391. SIAM, 2005.

[10] Lei Chang, Tengjiao Wang, Dongqing Yang, Hua Luan, and Shiwei Tang. Efficient algorithms for incremental maintenance of closed sequential patterns in large databases. *Data Knowl. Eng.*, 68(1):68–106, 2009.

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

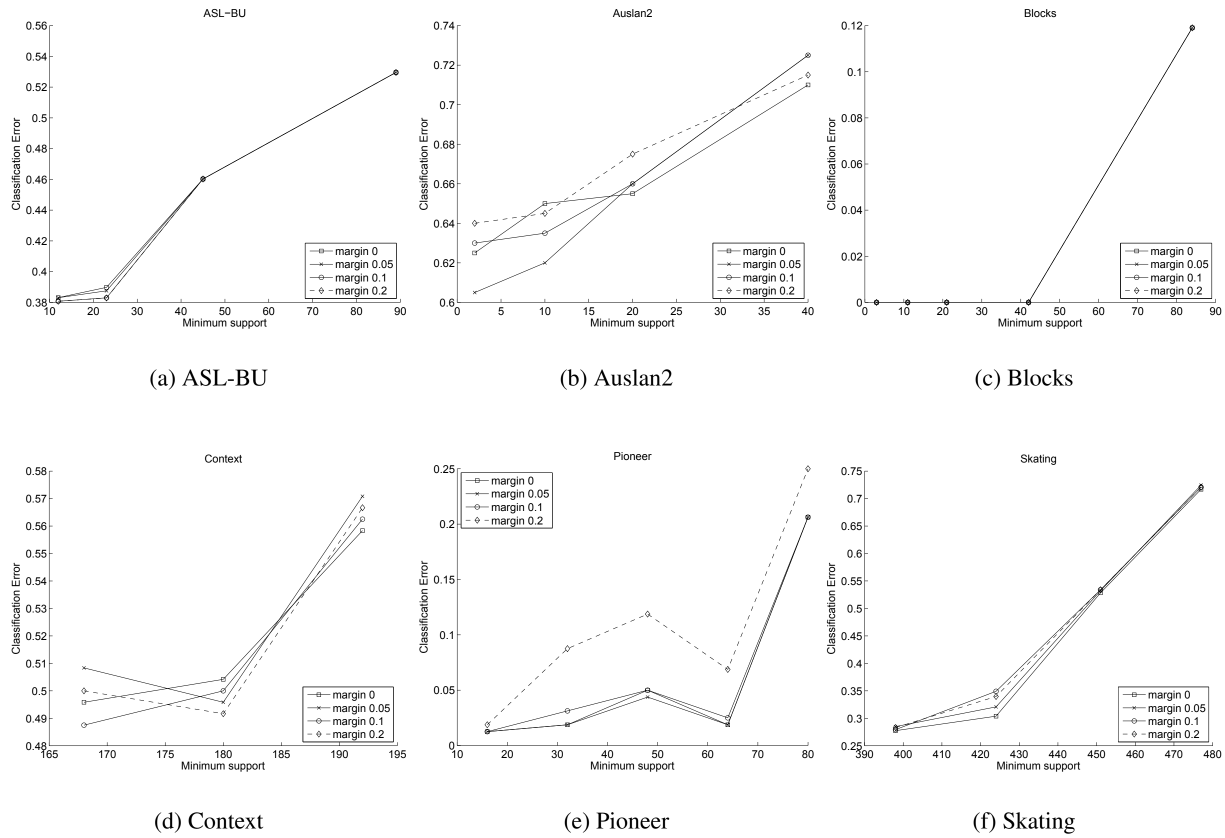
[12] J. Cheng, Y. Ke, and W. Ng. δ-tolerance closed frequent itemsets. In *Proc. 6th IEEE Int. Conf. on Data Mining*, pages 139–148. IEEE Press, 2006.

[13] L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 204–212. ACM, 2008.

[14] G. Dong and J. Pei. *Sequence Data Mining*. Morgan Kaufmann, 2007.

[15] A. Fern. *Learning Models and Formulas of a Temporal Event Logic*. PhD thesis, Purdue University, West Lafayette, IN, USA, 2004.

[16] R. Gupta, G. Fang, B. Field, M. Steinbach, and V. Kumar. Quantitative evaluation of approximate frequent pattern mining algorithms. In *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 301–309. ACM, 2008.



(a) ASL-BU

(b) Auslan2

(c) Blocks

(d) Context

(e) Pioneer

(f) Skating

Figure 2: SVM classification errors achieved with different minimum support thresholds and margin values.

[17] J. Han and M. Kamber. *Data Mining - Concepts and Techniques, 2nd edition*. Morgan Kaufmann, 2006.

[18] Frank Höppner. Discovery of temporal patterns - learning rules about the qualitative behaviour of time series. In *Proc. of the 5th European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD)*, pages 192–203. Springer, 2001.

[19] M. W. Kadous. *Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series*. PhD thesis, University of New South Wales, 2002.

[20] Arno J. Knobbe and Eric K. Y. Ho. Pattern teams. In *PKDD*, pages 577–584, 2006.

[21] C. Lucchese, S. Orlando, and R. Perego. Fast and memory efficient mining of frequent closed itemsets. *IEEE TKDE*, 18(1):21–36, 2006.

[22] H. Mannila, H. Toivonen, and I. Verkamo. Discovery of frequent episodes in event sequences. In *Proc. of the 1st Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 210–215. AAAI Press, 1995.

[23] J. Mäntyjärvi, J. Himberg, P. Kangas, U. Tuomela, and P. Huuskonen. Sensor signal data set for exploring context recognition of mobile devices. In *Proc. of 2nd Int. Conf. on Pervasive Computing (PERVASIVE 2004)*, pages 18–23. Springer, 2004.

[24] Fabian Moerchen, Michael Thies, and Alfred Ultsch. Efficient mining of all margin-closed itemsets with applications in temporal knowledge discovery and classification by compression. *Under review in Knowledge and Information Systems*.

[25] F. Mörchen. Algorithms for time series knowledge mining. In *Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 668–673. ACM Press, 2006.

[26] F. Mörchen. *Time Series Knowledge Mining*. PhD thesis, Philipps-University Marburg, Germany, 2006.

[27] F. Mörchen and A. Ultsch. Efficient mining of understandable patterns from multivariate interval time series. *Data Min. Knowl. Discov.*, 2007.

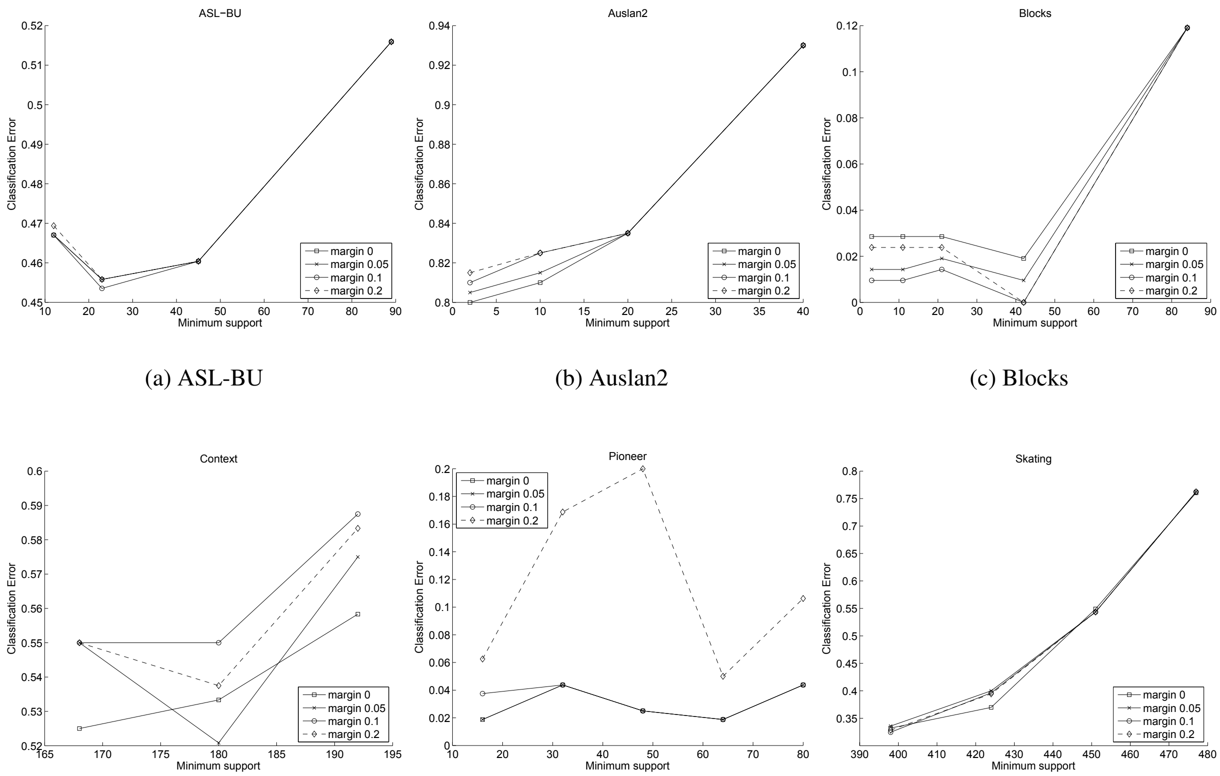
[28] Fabian Mörchen. Unsupervised pattern mining from symbolic temporal data. *SIGKDD Explor. Newsl.*, 9(1):41–55, 2007.

[29] Fabian Mörchen and Dmitriy Fradkin. Robust mining of time intervals with semi-interval partial order patterns. In *Proceedings of SIAM Conference on Data Mining (SDM)*, Columbus, Ohio, USA, 2010.

[30] P. Papaterou, G. Kollios, S. Sclaroff, and D. Gunopoulos. Discovering frequent arrangements of temporal intervals. In *Proc. of the 5th IEEE Intl. Conf. on Data Mining (ICDM)*, pages 354–361, 2005.

[31] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceeding of the 7th Intl. Conf. on Database Theory (ICDT)*, pages 398–416. Springer, 1999.

[32] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent



(a) ASL-BU (b) Auslan2 (c) Blocks

(d) Context (e) Pioneer (f) Skating

Figure 3: J48 classification errors achieved with different minimum support thresholds and margin values.

itemsets with convertible constraints. In *Proc. IEEE Intl. Conf. on Data Engineering*. IEEE, 2001.

[33] Jian Pei, Haixun Wang, Jian Liu, Ke Wang, Jianyong Wang, and Philip S. Yu. Discovering frequent closed partial orders from strings. *IEEE TKDE*, 18(11):1467–1481, 2006.

[34] Marc Plantevit and Bruno Crémilleux. Condensed representation of sequential patterns according to frequency-based measures. In *IDA '09: Proceedings of the 8th International Symposium on Intelligent Data Analysis*, pages 155–166, Berlin, Heidelberg, 2009. Springer-Verlag.

[35] C. Raïssi, T. Calders, and P. Poncelet. Mining conjunctive sequential patterns. *Data Min. Knowl. Discov.*, 17(1):77–93, 2008.

[36] N. Tatti and J. Vreeken. Finding good itemsets by packing data. In *Proc. 8th IEEE Int. Conf. on Data Mining*, 2008.

[37] M. van Leeuwen, J. Vreeken, and A. Siebes. Compression picks item sets that matter. In *Proc. European Conf. on Principles and Practice of Knowledge Discovery in Databases*, pages 585–592, 2006.

[38] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In *Proc. ICDE*, pages 79–90. IEEE Press, 2004.

[39] Jianyong Wang, Jiawei Han, and Chun Li. Frequent closed sequence mining without candidate maintenance. *IEEE Transactions on Knowledge and Data Engineering*, 19(8):1042–1056, 2007.

[40] Tao Wang. Compressing the set of frequent sequential patterns. In *FSKD '08: Proceedings of the 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, pages 330–334, Washington, DC, USA, 2008. IEEE Computer Society.

[41] Shin-Yu Wu and Yen-Liang Chen. Mining nonambiguous temporal patterns for interval-based events. *IEEE TKDE*, 19(6):742–758, 2007.

[42] M.J. Zaki. Mining non-redundant association rules. *Data Mining and Knowledge Discovery*, 9(3):223–248, 2004.

[43] Q. Zhao and S.S. Bhowmick. Sequential pattern mining: A survey. Technical report, Nanyang Technichal University, Singapore, 2003.

[44] Feida Zhu, Xifeng Yan, Jiawei Han, and Philip S. Yu. Efficient discovery of frequent approximate sequential patterns. In *ICDM '07: Proceedings of the Seventh IEEE International Conference on Data Mining*, pages 751–756, Washington, DC, USA, 2007. IEEE Computer Society.