Constraint-Based Sequential Pattern Mining with Decision Diagrams*

Amin Hosseininasab,[1] Willem-Jan van Hoeve,[1] Andre A. Cire[2]
[1]Tepper School of Business, Carnegie Mellon University, USA
[2]Dept. of Management, University of Toronto Scarborough, Canada
aminh@andrew.cmu.edu, vanhoeve@andrew.cmu.edu, andre.cire@rotman.utoronto.ca

Abstract

Constraint-based sequential pattern mining aims at identifying frequent patterns on a sequential database of items while observing constraints defined over the item attributes. We introduce novel techniques for constraint-based sequential pattern mining that rely on a multi-valued decision diagram (MDD) representation of the database. Specifically, our representation can accommodate multiple item attributes and various constraint types, including a number of non-monotone constraints. To evaluate the applicability of our approach, we develop an MDD-based prefix-projection algorithm and compare its performance against a typical generate-and-check variant, as well as a state-of-the-art constraint-based sequential pattern mining algorithm. Results show that our approach is competitive with or superior to these other methods in terms of scalability and efficiency.

Introduction

Sequential Pattern Mining (SPM) is a fundamental data mining task with a large array of applications in marketing, health care, finance, and bioinformatics, to name a few. Frequent patterns are used, e.g., to extract knowledge from data within decision support tools, to develop novel association rules, and to design more effective recommender systems. We refer the reader to (Fournier-Viger et al. 2017) for a recent and thorough review of SPM and its applications.

In practice, mining the entire set of frequent patterns in a database is not of interest, as the resulting number of items is typically large and may provide no significant insight to the user. It is hence desirable to restrict the mining algorithm search to smaller subsets of patterns that satisfy problem-specific constraints. For example, in online retail click-stream analysis, we may seek frequent browsing patterns from sessions where users spend at least a minimum amount of time on certain items that have specific price ranges. Such constraints limit the output of SPM and are much more effective in knowledge discovery, as compared to an arbitrary large set of frequent click-streams.

A naïve approach to impose constraints in SPM is to first collect all unconstrained frequent patterns, and then to apply a post-processing step to retain patterns that satisfy the desired constraints. This approach, however, may be expensive in terms of memory requirements and computational time, in particular when the resulting subset of constrained patterns is small in comparison to the full unconstrained set. Constraint-based sequential pattern mining (CSPM) aims at providing more efficient methods by embedding constraint reasoning within existing mining algorithms (Pei, Han, and Wang 2007; Negrevergne and Guns 2015). Nonetheless, while certain constraint types are relatively easy to incorporate in a mining algorithm, others of practical use are still challenging to handle in a general and effective way. This is particularly the case of non-monotone constraints representing, e.g., sums and averages of attributes.

Contributions. In this paper, we propose a novel representation of sequential database using a multi-valued decision diagram (MDD), a graphical model that compactly encodes the sequence of items and their attributes by leveraging symmetry. The MDD representation can be augmented with constraint-specific information, so that constraint satisfaction is either guaranteed or enforced during the mining algorithm. Finally, as a proof of concept, we implement a general prefix-projection algorithm equipped with an MDD to enforce several constraint types, including complex constraints such as average ("avg") and median ("md"). To the best of our knowledge, this paper is the first to consider the "sum," "avg," and "md" constraints with arbitrary item-attribute association within the pattern mining algorithm. Lastly, we provide an experimental comparison on real-world benchmark databases, and show that our approach is competitive with or superior to a state-of-the-art CSPM algorithm.

Related work

Research in CSPM has primarily focused on exploiting special properties of constraints, such as monotonicity or anti-monotonicity, to guarantee the feasibility of pattern extensions in the mining algorithm (Garofalakis, Rastogi, and Shim 1999; Zaki 2000; Lin and Lee 2005; Bonchi and Lucchese 2005; Chen and Hu 2006; Pei, Han, and Wang 2007; Nijssen and Zimmermann 2014; Mallick, Garg, and Grover 2014; Aoga, Guns, and Schaus 2017). Constraint types that do not possess such properties remain a challenge for CSPM algorithms, although some of these have been successfully incorporated in more general item-set mining on databases

*This work was supported by ONR grant N00014-18-1-2129.

Table 1: Example $\mathcal{SD}$, with attributes of time and price.

S_{ID}	Sequence: $\{(item, time, price)\}$
1	$\langle (B,1,5),(B,3,3)\rangle$
2	$\langle (B,3,3),(A,8,1),(B,9,3)\rangle$
3	$\langle (C,2,1),(C,5,2),(A,8,3)\rangle$

where events have no specific order (Soulet and Crémilleux 2005; Bistarelli and Bonchi 2007; Bonchi and Lucchese 2007; Le Bras, Lenca, and Lallich 2009; Leung et al. 2012), as well as in CSPM when items and attributes are interchangeable (Pei, Han, and Wang 2007).

Recently, constraint programming (CP) has emerged as a successful tool for CSPM (Negrevergne and Guns 2015; Kemmar et al. 2016; Kemmar et al. 2017; Aoga, Guns, and Schaus 2017; Guns et al. 2017). CP search techniques, albeit general, can potentially be more efficient when compared to specialized CSPM algorithms. Nonetheless, they still rely on constraint-specific properties to effectively prune undesired patterns.For example, (Aoga, Guns, and Schaus 2017) show how to effectively implement a number of prefix anti-monotone constraints into CP, but indicate that post-processing is still required to handle monotone constraints such as the minimum span.

Graphical representations of a database have been shown to be effective in item-set mining (Han et al. 2004; Pyun, Yun, and Ryu 2014; Borah and Nath 2018) and SPM (Masseglia, Poncelet, and Teisseire 2009). Previous works have also applied binary decision diagrams as a database modeling tool (Loekito and Bailey 2006; Loekito and Bailey 2007; Loekito, Bailey, and Pei 2010; Cambazard, Hadzic, and O'Sullivan 2010), which are effective when the sequences of the database are similar, but typically do not scale otherwise. We show that our MDD representation retains its size regardless of the similarity between sequences, and provides a more concise representation in the context of SPM.

Problem definition

We next formally describe the SPM problem and then discuss the handling of constraints.

The SPM database and mining algorithm

The SPM database consists of a set of events, which are modeled by a set of literals I denoted by *items*. Items $i \in I$ are associated with a set of *attributes* $\mathbb{A} = \{\mathcal{A}_1, ..., \mathcal{A}_{|\mathbb{A}|}\}$; for example, attributes can be price, quality, or time. A *sequence database* $\mathcal{SD}$ is defined as a collection of N item sequences $\{S_1, S_2, \ldots, S_N\}$, where all sequences are ordered with respect to the same attribute $\mathcal{A} \in \mathbb{A}$; e.g., occurrence in time. Table 1 illustrates an example $\mathcal{SD}$ with $N := 3$, $|I| := 3$, and $M := \max_{n\in\{1,\ldots,N\}} \{|S_n|\} = 3$, where items $i \in I$ are associated with time and price attributes.

The SPM task asks for the set of frequent *patterns* within $\mathcal{SD}$. A pattern $P = \langle i_1, i_2, \ldots, i_{|P|}\rangle$ is a subsequence of some $S \in \mathcal{SD}$. Let $S[j]$ denote the j^{th} position (i.e., item) of sequence S. A subsequence relation $P \preceq S$ holds if and only if there exists an embedding $e : e_1 \leq e_2 \leq ... \leq e_{|P|}$ such that $S[e_j] = i_j, i_j \in P$. For example, $P = \langle A, B\rangle$ is a subsequence of $S = \langle A, B, C, B\rangle$ with two possible embeddings $(1, 2)$ or $(1, 4)$. We define a super-sequence relation $S \succeq P$ analogously, with "$\leq$" replaced by "$\geq$". A pattern is frequent if it is a subsequence of at least θ number of sequences in $\mathcal{SD}$, where θ is a given frequency threshold.

The two best-known mining algorithms for SPM are the Apriori algorithm introduced by (Agrawal, Srikant, and others 1994), and the prefix-projection algorithm introduced by (Han et al. 2001). Both are iterative procedures and operate by extending frequent patterns one item at a time. In Apriori, candidate patterns are generated by expanding a pattern with all available items, and thereafter checking the frequency of generated candidates. As candidates may or may not be frequent, the Apriori algorithm suffers from the exponential explosion of the number of generated candidates and redundancy. The prefix-projection algorithm, in turn, operates by projecting each sequence $S \in \mathcal{SD}$ onto a smallest subsequence $S = \langle i_1, i_2, \ldots, i_j\rangle$, denoted by *prefix*, and searching for frequent items in this reduced database. Any sequence that is obtained by extending a frequent prefix is guaranteed to be frequent in the original database. Prefix-projection is more efficient than the Apriori algorithm as it rules out infrequent patterns more effectively, but it requires the full database to be in memory (Han et al. 2001).

Constraint satisfaction in CSPM

A constraint $C_{type}(\cdot)$ is a Boolean function imposed on either the patterns or their attributes. A pattern P satisfies a constraint if and only if $C_{type}(P) = true$. The objective of CSPM is to find all frequent patterns that satisfy a set of user-defined constraints. In particular, the challenge of CSPM is to impose constraints during the mining algorithm, rather than post-processing mined patterns for constraint satisfaction.

The standard framework for CSPM is to classify constraints as being monotone or anti-monotone, as such constraint are easy to handle within the mining algorithm (Pei, Han, and Wang 2007).[1] A constraint is *monotone* if its violation by a sequence S implies that all subsequences $\tilde{S} \preceq S$ also violate the constraint.

A constraint is *anti-monotone* if its violation by a sequence S implies violation by all super-sequences $\hat{S} \succeq S$. Table 2 lists common constraint types with their characterization. The concepts of monotonicity, anti-monotonicity, and violation are analogously extended to prefixes.

Constraints that are neither monotone nor anti-monotone are called *non-monotone* and are the most challenging to enforce during mining. While dedicated approaches have been developed for certain non-monotone constraints (Pei, Han, and Wang 2007), they are otherwise handled by post-processing (Aoga, Guns, and Schaus 2017). Our goal is to develop a generic platform to handle non-monotone constraints effectively.

[1] A third classification is succinctness, which allows immediate pattern generation using a formula rather than an algorithm.

Table 2: Characterization of constraints as monotone (M), anti-monotone (AM), or non-monotone (NM) for SPM.

Name	Constraint := definition	M	AM	NM
Maximal	$C_{mxl}(P) := \nexists P' \in \mathcal{SD} : P \prec P'$	•		
Sup-Patt	$C_{spt}(P) := \exists P' \in \mathcal{SD} : P' \prec P$	•		
Length	$C_{len}(P) \geq c := \lvert P \rvert \geq c$	•		
	$C_{len}(P) \leq c$		•	
Reg Expr	$C_{reg}(P) := P[i] \in \bar{I} \subset I$		⋆	
Gap	$C_{gap}(\mathcal{A}) \leq c := \alpha_j - \alpha_{j-1} \leq c,$ $\alpha_j \in \mathcal{A}, 2 \leq j \leq \lvert P \rvert$		⋆	
	$C_{gap}(\mathcal{A}) \geq c$		•	
Span	$C_{spn}(\mathcal{A}) \leq c := \max\{\mathcal{A}\} - \min\{\mathcal{A}\} \leq c$		•	
	$C_{spn}(\mathcal{A}) \geq c$	•		
Max/Min	$C_{max}(\mathcal{A}) \geq c, C_{min}(\mathcal{A}) \leq c$	•		
	$C_{max}(\mathcal{A}) \leq c, C_{min}(\mathcal{A}) \geq c$		•	
Stats	$C_{sum}(\mathcal{A}), C_{avg}(\mathcal{A}), C_{var}(\mathcal{A}), C_{med}(\mathcal{A})$			•

⋆Not anti-monotone, but prefix anti-monotone.

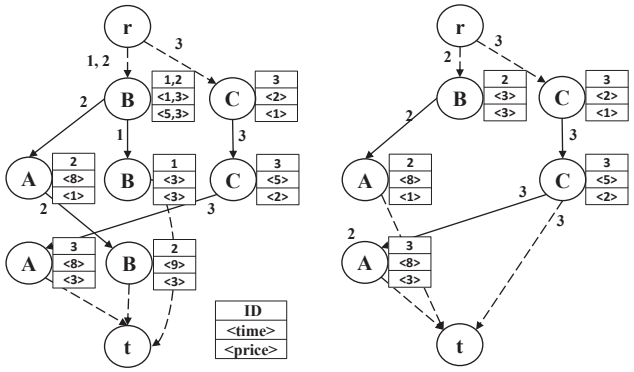

Figure 1: MDD database for the example SD in Table 1. Arcs skipping layers are not shown for clarity.

An MDD representation for $\mathcal{SD}$

MDDs are widely applied as an efficient data structure in verification problems (Wegener 2000) and were more recently introduced as a tool for discrete optimization and constraint programming (Bergman et al. 2016). Here, we use an MDD to fully encode the sequences from $\mathcal{SD}$; we refer to such data structure as an *MDD database*. We show how constraint satisfaction is achieved by storing constraint-specific information at the MDD nodes, thereby removing the need to impose constraint-specific rules in a mining algorithm.

MDD construction for the SPM problem

An MDD $M = (U, A)$ is a layered directed acyclic graph, where U is the set of nodes, and A is the set of arcs. Set U is partitioned into layers $(l_0, l_1, ..., l_{m+1})$, such that layers $l_i : 1 \leq i \leq m$ correspond to position (item) i of a sequence $S \in \mathcal{SD}$. Layers l_0, and l_{m+1} consist of single nodes, namely the root node $r \in l_0$, and the terminal node $t \in l_{m+1}$. The root and terminal node are used to model the start and end of all sequences, respectively. Figure 1.a shows the MDD database model for the $\mathcal{SD}$ of Table 1.

Layers $l_j, 1 \leq j \leq m$, contain one node per item $i \in I : \exists S \in \mathcal{SD}, S[j] = i$, and model the possible items at position j of all sequences $S \in \mathcal{SD}$. For example, layer 1 of the MDD database in Figure 1.a has two nodes corresponding to items B, C, and no node associated to item A. To distinguish which nodes are associated to which sequences $S \in \mathcal{SD}$, we define labels d_u for nodes $u \in U$, and store the associated sequence index S_{ID} in d_u. The first label of node B at layer 1 of Figure 1.a, indicates that sequences 1 and 2 contain item B at their first position. In addition, we store the attribute labels associated with the item, one per S_{ID} at each node. For example, in Figure 1.a we store the time and price attributes.

An arc $a = (u, v) \in A$, is directed from a node $u \in l_j$ to a node $v \in l_{j'} : j' > j$, and represents the next possible item after node u, for all sequences in $\mathcal{SD}$. Similar to nodes $u \in U$, labels d_a are defined for arcs $a \in A$ and store their associated sequences. A sequence S is thus represented by a path from r to t, following the nodes and arcs associated to S_{ID}. As we will search the MDD for patterns during the mining algorithm, we explicitly allow arcs to skip layers. That is, arc $(u, v) \in A$ can refer to any pair of nodes u, v on an r-t path P representing a sequence S. In Fig. 1 we only depict the arcs that represent the original sequences in $\mathcal{SD}$, for clarity. For example, the arc between node B at layer 1 and node B at layer 3 (following sequence $S_{\text{ID}} = 2$) is formally defined but omitted from the picture. Observe that any prefix or subsequence is represented by a partial path in the MDD, possibly using the arcs that skip layers. Lastly, we note that the MDD database (without imposed constraints) is built by a single scan of the database.

Imposing constraints on the MDD database

We use the MDD structure to enforce certain constraints on the MDD database itself. This has three main benefits, as follows. First, constraint satisfaction is performed only once, and not once per projected database as in the prefix-projection algorithm. Second, several constraints can be considered simultaneously, as opposed to iterative methods that consider each constraint individually and incur larger computational costs. Lastly, imposing constraints results in a smaller MDD, and consequently reduced computational requirements for the mining algorithm.

A constraint C_{type} can be imposed directly on the MDD if it is prefix monotone or prefix anti-monotone. That is, the feasibility of extending a pattern P ending at item i by an item i', is only dependent on the relationship between consecutive items i, i'. Examples of such constraints are the gap and regular expression constraints. An infeasible extension of such constraints is prevented by not creating an arc between their respective nodes. For example, if item i cannot be followed by item i', then no arc of the MDD database is constructed between their corresponding nodes.

Constraints on the MDD database are incorporated during its construction. In particular, the MDD database is built in increments using a backwards induction on the position j of a sequences $S \in \mathcal{SD}$. A backwards induction is chosen, as it allows us to gather constraint-specific information, used for constraint satisfaction later in the mining algorithm. For sequence S, the algorithm starts from the node corre-

sponding to the item at position $S[j] : j = |S|$, and checks whether this item may be used to extend a pattern ending in any of the sequence's previous items $i \in l_{j'} < l_j$. Whenever an extension is feasible, an arc (u, v) is created between the items' respective nodes in the MDD. The algorithm then increments and repeats the same procedure for the item in position $j - 1$.

By the construction above, a node connects to all nodes representing a feasible extension with respect to the imposed constraints. Thus, the mining algorithm needs only to search the children of a node $u \in U$ to extend any pattern ending at u. Figure 1.b shows an example of imposing constraint $C_{gap}(time) \geq 3$ on the MDD database of Figure 1.a.

Imposing constraints on the MDD database can be made more efficient by exploiting their properties such as anti-monotonicity. For example, given an anti-monotone constraint, if the extension of item i at $S[j]$ to an item at position $S[j']$ is infeasible, it is guaranteed that any extension of i to items $S[k] : k \geq j'$ is also infeasible. If a constraint is non-monotone, we are required to check its satisfaction for all possible extensions, which is done only if all monotone and anti-monotone constraints are satisfied.

Not all constraints can be imposed on the MDD database. The satisfaction of such constraints is performed during the mining algorithm, discussed in the next section.

Pattern mining with MDD databases

In this section, we discuss how to perform constraint reasoning by incorporating specific information into the MDD nodes. Such information is used to establish conditions to efficiently remove infeasible patterns from the database.

Information exploitation for effective mining

By construction, an r-u path in the MDD database represents the prefix of a pattern ending at node u. Similarly, any extension of this prefix is modeled by a u-t path. Post-processing patterns for constraint satisfaction corresponds to checking the feasibility of all u-t paths. We can, however, exploit the MDD structure to determine whether it is possible to extend an infeasible pattern to a feasible one. This is achieved by augmenting the MDD nodes with constraint-specific information that allow us to perform such reasoning.

For instance, consider a constraint $C_{min}(price) \geq 5$ and the extension of an infeasible pattern ending at node u, as shown in Figure 2. Observe that only one u-t path results in a feasible pattern. Instead of explicitly searching all u-t paths, we can store the minimum price reachable from nodes $u \in U$, during the MDD construction, and then use it to guarantee that a feasible extension exists.

Categories of constraint-specific information

We now describe constraint-specific information for a number of practical constraint classes. We only present the proof for lower bound constraints; upper bound conditions can be established analogously. We define $\alpha^u \in \mathcal{A}$ to be the attribute value of item i at node u of the MDD.

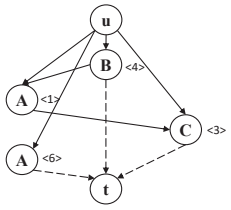

Figure 2: Extending a pattern ending at node u, with constraint $C_{min}(\mathcal{A}) \geq 5$. The label at each node represents the attribute of the item.

Span constraint: Let β_1^u and β_2^u denote the minimum and maximum values of α reachable from u, respectively. Values β_1^u are initially set to α^u. When adding an arc (u, v), we update $\beta_1^u \leftarrow \beta_1^v$ if $\beta_1^u > \beta_1^v$, and $\beta_2^u \leftarrow \beta_2^v$ if $\beta_2^u < \beta_2^v$ for node v. By this procedure, β_1^u, β_2^u give the minimum and maximum values of α reachable from u. Proposition 1 proves that by using these variables, we can guarantee the satisfaction of the span constraint.

Proposition 1. *An infeasible pattern P can be extended to a feasible pattern with respect to $C_{spn}(\alpha) \geq c$ if and only if*

$$\max\left\{\max_{\alpha\in P}\{\alpha\}, \beta_2^u\right\} - \min\left\{\min_{\alpha\in P}\{\alpha\}, \beta_1^u\right\} \geq c.$$

Proof. The necessity is straightforward. For the converse, assume $\alpha^{\max} - \alpha^{\min} < c$. Then no u-t path contains values of α such that P can become feasible. □

Sum constraint: Let β^u denote the maximum sum of values α reachable from u. We first initialize $\beta^u \leftarrow \alpha^u$. Next, when adding an arc (u, v), we update $\beta^u \leftarrow \beta^v + \alpha^u$ if $\beta^u < \beta^v + \alpha^u$, which results in the maximum sum possible to be stored for node u. Proposition 2 proves that this information is sufficient.

Proposition 2. *There exists a feasible extension from node u with respect to each individual constraint if and only if* $\sum_{\alpha\in P} \alpha + \beta^u \geq c$.

Proof. The necessity is straightforward. For the converse, assume $\sum_{\alpha\in P} \alpha + \beta^u < c$. By the construction of β^u, we can conclude $\sum_{\alpha\in P} \alpha + \sum_{\alpha\in(u,t)} \alpha < c$, for all u- paths. □

Average constraint: Let β_1^u denote a sum of values α on a u-t path, and β_2^u denote the number of attributes α contributing to the sum in β_1^u. For constraint $C_{avg}(\alpha) \geq c$, and any pattern P ending at node u, our objective is to generate values of β_1^u, β_2^u that give the maximum possible average $\frac{\sum_{\alpha\in P} \alpha+\beta_1^u}{|P|+\beta_2^u}$ above the threshold c.

The generation of β_1^u depends on the value c of constraint $C_{avg}(\alpha) \geq c$. Initially we set $\beta_1^u = \alpha^u$, and $\beta_2^u = 1$. When adding an arc (u, v) during the construction of the MDD, we update $\beta_1^u \leftarrow \alpha^u + \beta_1^v, \beta_2^u \leftarrow \beta_2^v + 1$ if $(\alpha^u + \beta_1^v) - c\,(1 + \beta_2^v) > \beta_1^u - c\beta_2^u$. This ensures that the best values to maximize $\frac{\sum_{\alpha\in P} \alpha+\beta_1^u}{|P|+\beta_2^u}$ are generated, proven in Lemma 3.

Lemma 3. *For constraint $C_{avg}(\alpha) \geq c$, the update procedure above generates values β_1^u, β_2^u that give the maximum average $\frac{\sum_{\alpha \in P} \alpha + \beta_1^u}{|P| + \beta_2^u}$ above threshold c, for a pattern p ending at node u.*

Proof. Proof by induction. By the initial definitions of β_1^u, β_2^u, the statement is true for any u in the last layer l_m of the MDD. Now assume the statement holds for all nodes in layer greater than l_j. For nodes u in layer l_j we choose the path giving the maximum average $\max_{u\text{-}t} \left\{ \frac{\sum_{\alpha \in P} \alpha + \beta_1^v + \alpha^u}{|P| + \beta_2^v + 1} - c \right\} = \max_{u\text{-}t} \{\beta_1^v + \alpha - c\,(\beta_2^v + 1)\}$. □

Proposition 4 shows that β_i^1, β_u^2 are the only required information to check satisfaction of the maximum average constraint. The proof for the minimum average constraint is similar, and omitted for brevity.

Proposition 4. *It suffices to record β_1^u, β_2^u as defined above, to check satisfaction for the minimum average constraint $C_{avg}(\alpha) \geq c$.*

Proof. The maximum average reachable from node u is $\frac{\beta_1^u}{\beta_2^u}$ by definition. Therefore, if $\frac{\sum_{\alpha \in P} \alpha + \beta_1^u}{|P| + \beta_2^u} < c$, then no (u, t) paths exists that satisfies $C_{avg}(\alpha) \geq c$ for a pattern ending at node u. □

Median constraint: Let β_1^u denote the maximum difference of the number of values $\alpha \geq c$ and the number of values $\alpha < c$, between all possible paths u-t, i.e. $\beta_1^u = \max_{u\text{-}t} \left\{ \left| \{\alpha \in u\text{-}t : \alpha \geq c\} \right| - \left| \{\alpha \in u\text{-}t : \alpha < c\} \right| \right\}$. Further, let β_2^u denote the maximum of values $\alpha < c$ contributing to the count in β_1^u, and β_3^u denote the minimum of values $\alpha \geq c$ contributing to the count in β_1^u. Observe that the satisfaction of $C_{med}(\alpha) \geq c$ can be determined using values β_1^u to β_3^u. Namely, if $\beta_1^u > 0$ then there exists more values α above c than below it, guaranteeing satisfaction. Similarly if $\beta_1^u < 0$ the median constraint is violated. If $\beta_1^u = 0$ then we calculate the average $\frac{\beta_2^u + \beta_3^u}{2}$ which gives the median.

The generation of β_1^u to β_3^u depends on the constant c. Initially, we set $\beta_1^u = 0, \beta_2^u = \min_{\alpha \in S} \{\alpha\} - 1, \beta_3^u = \alpha^u$ for all nodes $u : \alpha^u \geq c$, and $\beta_1^u = 0, \beta_2^u = \alpha^u, \beta_3^u = \max_{\alpha \in S} \{\alpha\} + 1$ for all remaining nodes. Next, during the construction of the MDD, for a node u, we find the path u-t that has the highest potential to extend an infeasible pattern P ending at u to a feasible one. The best path u-v-t, denote v-t, is a path that contains a feasible extension for P given any other feasible extensions available by the remaining u-v'-t paths, denote v'-t. We prove four dominance rules that when satisfied, guarantee this for v-t.

The first rule is if $\beta_1^v > \beta_1^{v'}$, proven valid in Lemma 5.

Lemma 5. *If $\beta_1^v > \beta_1^{v'}$ holds, and extension of a pattern P by path v'-t is feasible, so is the extension of P by path v-t.*

Proof. Let β_1^p denote the difference of the number of values $\alpha \in P : \alpha \geq c$ to the number of values $\alpha \in P : \alpha < c$. Then, $\beta_1^p + \beta_1^v > \beta_1^p + \beta_1^{v'}$, meaning there is a greater number of values $\alpha \geq c$ on path v-t, compared to path v'-t. □

All other conditions require $\beta_1^v = \beta_1^{v'}$. For these conditions, we first calculate $med_{v'} = \frac{\beta_2^{v'} + \beta_3^{v'}}{2}, med_v = \frac{\beta_2^v + \min\{\beta_3^v, \alpha^v\}}{2}$. Conditions two to four are proved in 6.

Lemma 6. *Given $\beta_1^v = \beta_1^{v'}$, any feasible extension of an infeasible pattern P by path v'-t is also feasible for path v-t, if one of the following three conditions hold: 1. $med_v \geq c, med_{v'} < c$, 2. $med_v \geq c, med_{v'} \geq c, \beta_2^v > \beta_2^{v'}$, 3. $med_v < c, med_{v'} < c, \beta_3^v > \beta_3^{v'}$.*

Proof. Let β_1^p to β_3^p be defined as before. For condition 1, as $med_{v'} < c$ and P is infeasible, any extension of P by u-t must have $\beta_1^p + \beta_1^{v'} > 0$, which is also satisfied by path v-t. For condition 2, if an infeasible pattern P can be extended to a feasible pattern by v'-t, then either $\beta_1^p + \beta_1^{v'} > 0$ which implies feasibility of v-t, or $\beta_1^p + \beta_1^{v'} = 0$. In this case, the only value of $\frac{\max\left\{\beta_2^p, \beta_2^{v'}\right\} + \min\left\{\beta_3^p, \beta_3^{v'}\right\}}{2}$ (i.e., the median of pattern P extended by v'-t), which is not guaranteed to be feasible or infeaisble is $\frac{\beta_2^{v'} + \beta_3^p}{2}$. However, if $\frac{\beta_2^{v'} + \beta_3^p}{2} \geq c$, we also have $\frac{\beta_2^v + \beta_3^p}{2} \geq c$. The proof of the third rule is similar to the second rule, and ommited due to space limits. □

If any of the above rules are satisfied, we update $\beta_1^u \leftarrow \beta_1^v + 1, \beta_2^u \leftarrow \max\{\beta_2^v, \alpha\}, \beta_3^u \leftarrow \beta_3^v$ if $\alpha^u \geq c$, or $\beta_1^u \leftarrow \beta_1^v - 1, \beta_2^u \leftarrow \beta_2^v, \beta_3^u \leftarrow \max\{\beta_3^u, \beta_3^v\}$ otherwise. Proposition 7 shows that these values are sufficient to determine whether an infeasible pattern P can be extended to a feasible one.

Proposition 7. *Let $\beta_1^p - \beta_3^p$ be defined as before. There exists a feasible extension from node u with respect to $C_{med}(\alpha) \geq c$ if and only if $\beta_1^u + \beta_1^p > 0$, or $\beta_1^u + \beta_1^p = 0, \frac{\min\left\{\beta_3^p, \beta_3^p\right\} + \max\left\{\beta_2^p, \beta_2^u\right\}}{2} \geq c$.*

Proof. The necessity is straightforward. For the converse, first assume $\beta_1^p + \beta_1^v < 0$, then by Lemma 6, no u-t path contains enough values $\alpha \geq c$ to satisfy $C_{med}(\alpha) \geq c$. For the second condition, if $\beta_1^u + \beta_1^p = 0, \frac{\min\left\{\beta_3^p, \beta_3^u\right\} + \max\left\{\beta_2^p, \beta_2^u\right\}}{2} < c$, then by Lemma 6, the maximum median between all u-t paths is below threshold c. □

Mining the MDD database with prefix-projection

We now present our *MDD prefix-projection* (MPP) algorithm, which performs prefix-projection on the MDD database. The first step of the algorithm is to find all frequent items i, i.e. patterns of size one, using a depth-first-search. This is automatically done during the construction of the MDD database, and modeled by the children of root node r. In the next steps, the algorithm attempts to expand a

frequent pattern generated in previous iterations. Using the stored information in the MDD, we prune extensions that cannot lead to a feasible pattern. In particular, for an infeasible pattern P ending at node $u \in U$, the algorithm uses the information stored at u to determine whether P may be extended to a feasible pattern. If a feasible extension does not exist, the search is pruned. Otherwise, pattern P is extended and investigated in future iterations.

In contrast to searching the database rows in prefix-projection, the MPP algorithm follows feasible paths in the MDD database. This leads to a more efficient search, as some infeasible extensions have been removed when constructing the MDD database. The trade-off is that finding paths corresponding to a sequence S requires a search on labels d_u, a_u, thereby incurring additional computational cost. For efficient memory utilization, the MDD is not physically projected, but rather *pseudo projected* (Han et al. 2001). In pseudo projection, only the initial $\mathcal{SD}$ is stored in memory, and search is initiated from "projection pointers" pointing to the MDD nodes.

In prefix-projection, all N sequences are searched in each iteration, and an item $i \in I$ is frequent if its final count is at least θ. As opposed to searching all N sequences, we propose to stop when it is guaranteed that an item i is not frequent. Let n denote the number of sequences searched so far when searching for i, and let $Sup(P)$ denote the number of sequences that contain pattern P. We use the following proposition to detect that item i cannot be frequent, given a frequent pattern P:

Proposition 8. *If $n - Sup(i) > Sup(P) - \theta$, item i cannot be frequent in the projected database.*

Proof. The left-hand-side is the number of searched sequences that do not contain i, and the right-hand-side is the maximum number of sequences that do not contain i while it remains frequent. □

Projecting the minimal prefix containing a pattern P (as done in SPM) is not sufficient for CSPM (Aoga, Guns, and Schaus 2017). Extensions from the minimal prefix may violate a constraint, while it may be the case that another larger prefix of the sequence satisfies such extensions. For example, the minimal prefix containing item C in sequence 3 of Table 1 cannot be extended by item A under a constraint $C_{gap}(time) \leq 3$. However, extending the larger prefix containing C is feasible. We are thus required to store all prefixes and their extension at each iteration of MPP.

A time-consuming task of the general prefix-projection algorithm is to determine whether a specific item i exists in sequences of the projected database. To avoid searching the entire sequence for every item, (Aoga, Guns, and Schaus 2017) store the last position of items $i \in I$ for sequences $S \in \mathcal{SD}$. An MDD database enables search for the extension of all items $i \in I$ simultaneously, resulting in more efficient search. That is, as opposed to searching for a specific item i, all children of node u are searched, and record the items which enable a feasible extension.

Table 3: Five real-life datasets and their features.

$\mathcal{SD}$	N	$\|I\|$	M	$avg(\|S\|)$*
Kosarak	837,206	41,001	2,498	9.3
MSNBC	989,818	19	29,591	10.5
Kosarak (small)	59,261	20,894	796	9.2
BMSWebView1	26,667	497	267	4.4
BMSWebView2	52,619	3,335	161	6.3

*Average length of sequences

Numerical results

For our numerical tests, we use real-life click-stream benchmark databases[2], listed in Table 3. We note that two of these databases, Kosarak and MSNBC, are considerably larger than those typically reported in the CSPM literature, with about 900,000 sequences of length up to 29,500, and containing up to 40,000 items. None of these standard benchmark datasets are annotated with attributes. To be able to evaluate our approach, we therefore generate three attributes of time, price, and quality, as follows. For the time attribute, we randomly generate a number between 0 and 600 seconds, to represent the time spent by users at each click. With a probability of 5%, we model the user leaving the session by setting the time between clicks to a value between 1 to 10 hours. For the price and quality attributes, we generate a number between 1 and 100 for each item $i \in S, \forall S \in \mathcal{SD}$.

All algorithms are coded in C++, with the exception of PPICt which is coded in Scala.[3] All experiments are executed on the same PC with an Intel Xeon 2.33 GHz processor, 24GB of memory, using Ubuntu 12.04.5 as operating system. We limit all tests to use one core of the CPU. The MPP code is available and open source.[4]

Comparison with prefix-projection and constraint checks

Our first goal is to evaluate the impact of the MDD database and the associated constraint reasoning, especially in presence of more complex constraints. However, no other CSPM system accommodates constraints such as average and median and multiple item attributes. Because simple generate-and-test (via post-processing) does not scale due to the size of the databases, we developed a prefix-projection algorithm for the original database, that can handle multiple item attributes and effectively prune the search space for anti-monotone constraints such as gap and maximum span. We name this algorithm Prefix-Projection with Constraint Checks (PPCC). PPCC operates by prefix-projection and extends a pattern P if it satisfies all anti-monotone constraints, and prunes the extension otherwise. For non-monotone constraints, PPCC extends infeasible patterns with the hope that a feasible super-pattern exists, and performs a constraint check at the end.

[2] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

[3] We thank the developers of PPICt for sharing their code.

[4] https://github.com/aminhn/MPP

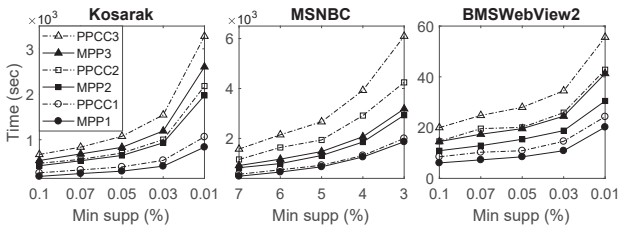

Figure 3: Mining with constraints $30 \leq C_{gap}(time) \leq 900, 900 \leq C_{spn}(time) \leq 3600, 30 \leq C_{avg}(price) \leq 70, 40 \leq C_{med}(price) \leq 60, 40 \leq C_{avg}(quality) \leq 60, 30 \leq C_{med}(quality) \leq 70$. Attributes and their corresponding constraints are added incrementally from 1 to 3.

In Figure 3 we compare the performance of MPP and PPCC in terms of total CPU time (MDD construction plus mining algorithm), given minimum support (Min supp) as a percentage of the total number of sequences. The experiment uses three scenarios with constraints on one, two, and three attributes, respectively:

time: $30 \leq C_{gap}(time) \leq 900,\ 900 \leq C_{spn}(time) \leq 3600,$
price: $30 \leq C_{avg}(price) \leq 70,\ 40 \leq C_{med}(price) \leq 60,$
quality: $40 \leq C_{avg}(quality) \leq 60,\ 30 \leq C_{med}(quality) \leq 70.$

Scenario one (PPCC1 and MPP1) only considers the time constraints. Scenario two (PPCC2 and MPP2) considers the time and price constraints. Scenario three (PPCC3 and MPP3) considers all time, price, and quality constraints. The results in Figure 3 show that mining more constrained patterns takes more time for both methods. However, MPP is always more efficient than PPCC, and often considerably. For example, finding all frequent patterns with minimum support of 4% with all constraints (scenario three) in the MSNBC database takes PPCC about 4,000s while MPP only needs about 2,000s. Moreover, Table 4 shows that the time required to construct the MDD database and generate constraint specific information is quite small. This indicates that our MDD database can be used to effectively and efficiently handle constraints such as average and median.

Comparison with PPICt

We next compare our approach to the state-of-the-art CSPM algorithm PPICt, which is implemented in the CP framework OscaR[5] (Aoga, Guns, and Schaus 2017). PPICt accommodates a wide range of constraints, including gap and maximum span constraints, but is restricted to a single attribute. We therefore evaluate MPP and PPICt for mining patterns with the following gap and maximum span constraints over the time attribute:

$$30 \leq C_{gap}(time) \leq 90,\ \ 900 \leq C_{spn}(time) \leq 3600.$$

Initial tests indicated that the PPICt code is unstable when executed on the full databases Kosarak and MSNBC. We therefore executed the codes on the smaller benchmark variant of Kosarak (which is also used in (Aoga, Guns, and Schaus 2017)), BMSWebView1, and BMSWebView2. The results are presented in Figure 4, which follows the same format as Figure 3.

[5]https://bitbucket.org/oscarlib/oscar/wiki/Home

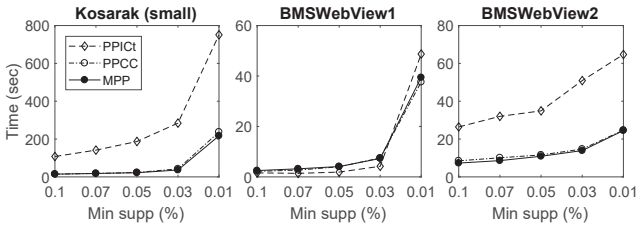

Figure 4: Mining with one item attribute (time) and constraints $30 \leq C_{gap}(time) \leq 900, C_{spn}(time) \leq 3600$.

Table 4: Time (in seconds) required for MDD construction and information generation.

Algorithm	Kosarak	MSNBC	BMS2	BMS1	Kosarak(small)
MPP1	47	45	2	-	-
MPP2	106	103	4	-	-
MPP3	151	158	5	-	-
MPP	-	-	2	0.5	4

A first observation is that MPP and PPCC produce almost identical results, as they both benefit from the same pruning rules for anti-monotone constraints. The time required to build the MDD database, shown in Table 4, is made up by a faster prefix-projection algorithm due to implementing the gap constraints on the MDD itself. Both MPP and PPCC also outperform PPICt on Kosarak (small) and BMSWebview2, but all three methods perform similarly on BMSWebView1. However, PPICt uses significantly more memory, up to 14Gb, while MPP uses up to 1Gb, and PPCC consumes the lowest with at most 0.5Gb. We conclude that on this benchmark our approach is competitive with or more efficient than PPICt.

Conclusion

In this paper, we developed a novel MDD representation for CSPM. We prove how constraint satisfaction is achieved for a number of constraints, including sum, average, and median, by storing constraint-specific information at the MDD nodes. Moreover, our approach is able to accommodate several item attributes with constraints, which occur frequently in real-world problems.

We embedded our MDD representation within a prefix-projection algorithm, called MPP, and performed an experimental evaluation on real-life benchmark databases with up to 980,000 sequences and 40,000 items. The results showed that the MPP mining algorithm is always more efficient than a prefix-projection algorithm with constraint checks. The benefits of MPP become larger as we increase the size of the database, the number of constraints, or the number of attributes. Although MPP is primarily designed for efficient constraint satisfaction of rich constraints and multiple item attributes, it remains competitive with a CP-based state-of-the-art CSPM algorithm, for databases with only one item attribute and anti-monotone constraints.

References

Agrawal, R.; Srikant, R.; et al. 1994. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, 487–499.

Aoga, J. O. R.; Guns, T.; and Schaus, P. 2017. Mining time-constrained sequential patterns with constraint programming. *Constraints* 1–23.

Bergman, D.; Cire, A. A.; van Hoeve, W.-J.; and Hooker, J. N. 2016. *Decision Diagrams for Optimization*. Springer.

Bistarelli, S., and Bonchi, F. 2007. Soft constraint based pattern mining. *Data & Knowledge Engineering* 62(1):118–137.

Bonchi, F., and Lucchese, C. 2005. Pushing tougher constraints in frequent pattern mining. In *PAKDD*, volume 5, 114–124. Springer.

Bonchi, F., and Lucchese, C. 2007. Extending the state-of-the-art of constraint-based pattern discovery. *Data & Knowledge Engineering* 60(2):377–399.

Borah, A., and Nath, B. 2018. Fp-tree and its variants: Towards solving the pattern mining challenges. In *Proceedings of First International Conference on Smart System, Innovations and Computing*, 535–543. Springer.

Cambazard, H.; Hadzic, T.; and O'Sullivan, B. 2010. Knowledge compilation for itemset mining. In *ECAI*, volume 10, 1109–1110.

Chen, Y.-L., and Hu, Y.-H. 2006. Constraint-based sequential pattern mining: The consideration of recency and compactness. *Decision Support Systems* 42(2):1203–1215.

Fournier-Viger, P.; Lin, J. C.-W.; Kiran, R. U.; Koh, Y. S.; and Thomas, R. 2017. A survey of sequential pattern mining. *Data Science and Pattern Recognition* 1(1):54–77.

Garofalakis, M. N.; Rastogi, R.; and Shim, K. 1999. Spirit: Sequential pattern mining with regular expression constraints. In *VLDB*, volume 99, 7–10.

Guns, T.; Dries, A.; Nijssen, S.; Tack, G.; and De Raedt, L. 2017. Miningzinc: A declarative framework for constraint-based mining. *Artificial Intelligence* 244:6–29.

Han, J.; Pei, J.; Mortazavi-Asl, B.; Pinto, H.; Chen, Q.; Dayal, U.; and Hsu, M. 2001. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *proceedings of the 17th international conference on data engineering*, 215–224.

Han, J.; Pei, J.; Yin, Y.; and Mao, R. 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery* 8(1):53–87.

Kemmar, A.; Loudni, S.; Lebbah, Y.; Boizumault, P.; and Charnois, T. 2016. A global constraint for mining sequential patterns with gap constraint. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 198–215. Springer.

Kemmar, A.; Lebbah, Y.; Loudni, S.; Boizumault, P.; and Charnois, T. 2017. Prefix-projection global constraint and top-k approach for sequential pattern mining. *Constraints* 22(2):265–306.

Le Bras, Y.; Lenca, P.; and Lallich, S. 2009. On optimal rule mining: A framework and a necessary and sufficient condition of antimonotonicity. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 705–712. Springer.

Leung, C. K.-S.; Jiang, F.; Sun, L.; and Wang, Y. 2012. A constrained frequent pattern mining system for handling aggregate constraints. In *Proceedings of the 16th International Database Engineering & Applications Sysmposium*, 14–23. ACM.

Lin, M.-Y., and Lee, S.-Y. 2005. Efficient mining of sequential patterns with time constraints by delimited pattern growth. *Knowledge and Information Systems* 7(4):499–514.

Loekito, E., and Bailey, J. 2006. Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 307–316. ACM.

Loekito, E., and Bailey, J. 2007. Are zero-suppressed binary decision diagrams good for mining frequent patterns in high dimensional datasets? In *Proceedings of the sixth Australasian conference on Data mining and analytics-Volume 70*, 139–150. Australian Computer Society, Inc.

Loekito, E.; Bailey, J.; and Pei, J. 2010. A binary decision diagram based approach for mining frequent subsequences. *Knowledge and Information Systems* 24(2):235–268.

Mallick, B.; Garg, D.; and Grover, P. S. 2014. Constraint-based sequential pattern mining: a pattern growth algorithm incorporating compactness, length and monetary. *Int. Arab J. Inf. Technol.* 11(1):33–42.

Masseglia, F.; Poncelet, P.; and Teisseire, M. 2009. Efficient mining of sequential patterns with time constraints: Reducing the combinations. *Expert Systems with Applications* 36(2):2677–2690.

Negrevergne, B., and Guns, T. 2015. Constraint-based sequence mining using constraint programming. In *Proceedings of CPAIOR*, volume 9075 of *LNCS*, 288–305. Springer.

Nijssen, S., and Zimmermann, A. 2014. Constraint-based pattern mining. In *Frequent pattern mining*. Springer. 147–163.

Pei, J.; Han, J.; and Wang, W. 2007. Constraint-based sequential pattern mining: the pattern-growth methods. *Journal of Intelligent Information Systems* 28(2):133–160.

Pyun, G.; Yun, U.; and Ryu, K. H. 2014. Efficient frequent pattern mining based on linear prefix tree. *Knowledge-Based Systems* 55:125–139.

Soulet, A., and Crémilleux, B. 2005. Exploiting virtual patterns for automatically pruning the search space. In *International Workshop on Knowledge Discovery in Inductive Databases*, 202–221. Springer.

Wegener, I. 2000. *Branching programs and binary decision diagrams: theory and applications*, volume 4. SIAM.

Zaki, M. J. 2000. Sequence mining in categorical domains: incorporating constraints. In *Proceedings of the ninth international conference on Information and knowledge management*, 422–429. ACM.