Constraining the Search Space in Temporal Pattern Mining

Andreas D. Lattner and Otthein Herzog
TZI – Center for Computing Technologies, Universität Bremen
PO Box 330 440, D-28334 Bremen, Germany
{adl|herzog}@tzi.de

Abstract

Agents in dynamic environments have to deal with complex situations including various temporal interrelations of actions and events. Discovering frequent patterns in such scenes can be useful in order to create prediction rules which can be used to predict future activities or situations. We present the algorithm MiTemP which learns frequent patterns based on a time interval-based relational representation. Additionally the problem has also been transfered to a pure relational association rule mining task which can be handled by WARMR. The two approaches are compared in a number of experiments. The experiments show the advantage of avoiding the creation of impossible or redundant patterns with MiTemP. While less patterns have to be explored on average with MiTemP more frequent patterns are found at an earlier refinement level.

1 Introduction

Agents in dynamic environments have to deal with complex situations including various temporal interrelations of actions and events. If more elaborated technologies like planning should be used, the representation of the agent’s belief including background knowledge for its behavior decision can become very complex, too. It is necessary to represent knowledge about object classes and their properties, actual scenes with objects, their attributes and relations. If even more complex scenes with temporal extents shall be described this additional dimension must also be incorporated in the formalism.

Discovering frequent patterns in dynamic scenes can be useful in order to create prediction rules which can be used to predict future activities of other agents or to predict future situations and, thus, adapt the own behavior by taking into account this additional knowledge. In previous work a relational representation with temporal validity intervals and algorithms for mining temporal patterns have been introduced in [Lattner et al., 2006]. Here, we present a new algorithm which is also based on such a representation but avoids the creation of redundant patterns by defining an optimal refinement operator similar to the one in [Lee, 2006]. Additionally to the own implementation the problem has also been transfered to a pure relational association rule mining task which can be handled by WARMR [Dehaspe and Toivonen, 1999].

2 Related Work

Association rule mining addresses the problem of discovering association rules in data. One typical example is the mining of rules in basket data [Agrawal et al., 1993]. Different algorithms have been developed for the mining of association rules in item sets (e.g., Apriori [Agrawal and Srikant, 1994]). [Mannila et al., 1997] extended association rule mining by taking event sequences into account. They describe algorithms which find all relevant episodes which occur frequently in the sequence. Höppner presents an approach for learning rules about temporal relationships between labeled time intervals [Höppner, 2003]. The time intervals consist of propositional events and temporal relations are described by Allen’s interval logic [Allen, 1983].

Dehaspe and Toivonen combine association rule mining algorithms with ILP techniques. Their system WARMR is an extension of Apriori for mining association rules over multiple relations [Dehaspe and Toivonen, 1999]. The generated rules consist of sets of logical atoms. This more expressive representations (compared to itemset mining) allows for discovering relational association rules.

[Kaminka et al., 2003] introduce an approach which creates a sequence of certain events or behaviors from objects’ positions in RoboCup soccer matches and searches for frequent sequences in the data. In their work they compare two approaches based on frequency counts and statistical dependencies. The events in the sequences do not have a temporal extent and the learned patterns do not abstract from concrete objects in the events.

[Lee, 2006] presents an approach to mine first-order logical (SeqLog) patterns from sequential relational data. He defines optimal refinement operators and algorithms for finding all frequent patterns. The support is defined by the number of event sequences that match the pattern.

The approaches of Höppner, Kaminka et al., and Lee are quite similar to the one presented here. In contrast to Höppner and Kaminka our approach can learn relational patterns with variables which is also supported by [Lee, 2006]. Like Höppner we take an interval-based representation as input and mine temporal patterns with temporal inter-relations between these intervals. We also use a similar support definition which is based on the probability to find a pattern at a random sliding window position in the sequence. In contrast to our work, Kaminka et al. and Lee’s approaches are based on event sequences without temporal extent.

Our approach combines and extends these existing approaches. To the best of our knowledge no approach has addressed the mining of frequent temporal patterns from multi-relational time interval-based data. Our approach allows for taking hierarchical class information into account.

while
(while existing approaches just provide types for variables). Reasoning techniques are used to exploit the knowledge about temporal relations and about classes in order to reduce the number of patterns to be generated and to avoid checking inconsistent patterns.

3 Definitions and Problem Statement

The goal of the mining task is to find the set of all frequent temporal patterns from a dynamic scene. Before the approach is described in detail we provide some definitions. Let \( V, O, C, \) and \( IR \) be the sets of variables, objects, classes, and temporal interval relations, respectively.

Definition 3.1 (Dynamic Scene) A dynamic scene is described by the 4-tuple \( ds = (P, O, i, DS) \) where \( P \) is the set of predicate instances, \( O \) is the set of objects in the dynamic scene, \( i : O \rightarrow C \) maps the objects to classes (instance-of relation), and \( DS \) is the dynamic scene schema.

Definition 3.2 (Dynamic Scene Schema) The schema of a dynamic scene \( DS = (C, sc, PD, IR) \) consists of all schematic information. \( C \) is the set of classes and \( sc : C \rightarrow C \) maps classes to their super classes and thus describes the class hierarchy. \( C \) consists of at least one element which denotes the most general class (object). \( PD \) is the set of predicate definitions and \( IR \) is the set of the temporal interval relations.

Predicate definitions consist of the identifier, the arity, and the allowed ranges for the objects in their instances.

Definition 3.3 (Predicate Definition) A predicate definition \( pd \) is defined as \( pd = (pd_{name}, pd_{arity}, pd_{classes}) \) with \( pd_{classes} = (c_1, c_2, \ldots, c_{pd_{arity}}) \). All \( c_i \) denote classes in the dynamic scene schema, i.e., \( c_i \in C \) with \( 1 \leq i \leq pd_{arity} \).

Definition 3.4 (Predicate Instance) Predicate instances \( pi = (pd, p_{objects}, (s, e)) \) are instances of predicate definition \( pd \), consist of a list of object identifiers \( p_{objects} = (o_1, o_2, \ldots, o_{pd_{arity}}) \) with \( \forall o_i : o_i \in O \) of the dynamic scene, and additionally contain an interval of validity \((s, e)\) with start time \( s \) and end time \( e \).

For a better understanding we denote predicate instances in a more readable way: \( holds(predicate(o_1, o_2, \ldots, o_{pd_{arity}}), (s, e)) \) represents a predicate with \( pd_{name} = predicate, p_{objects} = (o_1, o_2, \ldots, o_{pd_{arity}}) \), start time \( s \), and end time \( e \). An example for a predicate in this notation is: \( holds(inBallControl(p7), (17, 42)) \).

Definition 3.5 (Interval Relation Function) The interval relation function \( ir : (\mathbb{N}, \mathbb{N}) \times (\mathbb{N}, \mathbb{N}) \rightarrow IR \) maps time interval pairs to interval relations.

It depends on the used interval relations \( IR \) how the actual mapping from the interval pairs to the interval relation has to be performed. Using, for instance, Allen’s interval relations \( ir((s_1, e_1), (s_2, e_2)) = b \) (before) if (and only if) \( e_1 < s_2 \) [Allen, 1983].

An atomic pattern consists only of one predicate pattern. The difference to predicate instances is that the list of arguments do not need to denote objects. In the general case the elements of the pattern are variables that can be bound to objects while pattern matching. However, it is also allowed to have arguments bound to objects in the pattern already.

Definition 3.6 (Atomic Pattern) An atomic pattern is defined as \( p = (pd_{-name}, pd_{arity}) \) where \( pd \) denotes a predicate definition and \( pd_{arity} \) specifies a list of terms \( pd_{arity} = (v_1, v_2, \ldots, v_{pd_{arity}}) \). All \( v_i \) are either elements of \( O \) as defined in the dynamic scene or are elements of \( V \), i.e., they hold \( \forall v_i \in V \cup O \).

Definition 3.7 (Conjunctive Pattern) A conjunction of atomic patterns is called conjunctive pattern. It connects the atomic patterns by a conjunction (logical AND): \( p_1 \land p_2 \land \ldots \land p_n \) where the \( p_i \) are atomic patterns with \( 1 \leq i \leq n \); \( n \) is called the size of the pattern.

Similarly to the predicate instances above we introduce a short notation for conjunctive patterns: \( predicate_1(v_1, \ldots, v_{pd_{arity}}) \land \ldots \land predicate_n(v_1, \ldots, v_{pd_{arity}}) \). An example of a conjunctive pattern with two predicates is uncovered\((X) \land pass(Y, X)\).

Definition 3.8 (Class Restriction) The class restriction defines for each variable \( v_i \) of a conjunctive pattern its least general class \( c_i \). For a given variable list \( (v_1, v_2, \ldots, v_n) \) the class restriction is represented by a class list \( (c_1, c_2, \ldots, c_n) \).

Variable unifications define if certain variables in a (conjunctive) pattern should refer to the same object in the assignment during pattern matching, i.e., if variables are unified.

Definition 3.9 (Variable Unification) A variable unification of a pattern \( p \) is defined as the unification of two different arguments \( v_1 \) and \( v_2 \) of one or two predicates of \( p \), i.e., it must hold that \( v_1 = v_2 \).

Binding a variable to a constant (i.e., to an instance) is denoted as instantiation:

Definition 3.10 (Instantiation) A variable \( v_i \) is instantiated if it is bound to an instance of the set of objects in the dynamic scene, i.e., if \( v_i = o \) with \( o \in O \).

A temporal restriction defines the constraints w.r.t. the validity intervals of two predicates in a conjunctive pattern. The order of the predicates in a pattern defines a temporal order implicitly already. A predicate must have an earlier or identical start time as all its succeeding predicates. Therefore, we define \( IR_{\text{older}} \subseteq IR \) including those temporal relations where the start time of the first interval \( s_1 \) is before the start time of the second interval \( s_2 \), i.e., \( s_1 < s_2 \) and for the “head to head” temporal relations we define \( IR_{\text{=}} \subseteq IR \) where the start times are equal, i.e., \( s_1 = s_2 \).

Definition 3.11 (Temporal Restriction) The temporal restriction \( IR = \{ IR_{[1, 2], \ldots, IR_{[n-1, n]} \} \} \) of a conjunctive pattern \( p \) with size \( n \) is defined as the set of pairwise temporal relations between all predicates. For each predicate pair \( (pred_i, pred_j) \) of the pattern \( p \), where \( pred_i \) appears before \( pred_j \) in the pattern, i.e., \( i < j \), the possible temporal relations between these two intervals are defined by the set \( IR_{[i, j]} \). It must hold that \( \forall tr_{k} \in IR_{[i, j]} : tr_{k} \in IR_{\text{older}} \cup IR_{\text{=}} \) with \( 1 \leq i < n \) and \( i < j \leq n \) due to the implicit temporal order of the predicates. If the name \( pd_{name_i} \) of \( pred_i \) is smaller than \( pd_{name_j} \) of \( pred_j \), w.r.t. a lexicographic order it must hold that \( \forall tr_{k} \in IR_{[i, j]} : tr_{k} \in IR_{\text{older}} \) in order to have a canonical representation of the sequences.

In the experiments described in section 6 we use just five temporal relations which can be seen as a condensed subset.
of the temporal relations introduced by [Freksa, 1992] and
[Allen, 1983]: before and after (<, >), older & contemporary
and younger & contemporary (<, >), and head
to head (|=). Thus, in our case \( TR = \{<, <c|\mid |= >, >\} \),
\( TR_{old} = \{<, <, >\} \), and \( TR_{eq} = \{|=\} \). The motivation
for these temporal relations is due to keeping complexity
low and still having the relevant temporal relations for set-
ing up prediction rules. The composition table for these
temporal relations is shown in Table 1.

\( \forall \) stance is assigned more than once, i.e.:

Furthermore, it must hold that no predicate in-
responding (instantiated) predicate
\( p \) \( \) in the conjunction must be true (within a defined

patterns to a dynamic scene. Pattern matching is essential
and temporal patterns, we can define how to match such
patterns to a dynamic scene. Pattern matching is essential

as not all matches have to be collected or maybe even fur-
ther processed. The monotonicity property for this support
definition holds and the support intervals of previous steps
(i.e., of more general patterns) can be reused in order to
restrict the search to parts of the temporal sequence in the
subsequent levels.

Definition 3.14 (Support) Let \( p \) be a temporal pattern, \( ds \)
the dynamic scene, and \( M \) the set of matches. The validity
interval of a single match \( m_i \in M \) is defined as \( v_i = [s_{max} - w + 1, e_{min} + w] \) with \( s_{max} \),
being the maximal start time and \( e_{min} \), the minimal end time of all predicate
instances in \( m_i \). The support of \( p \) w.r.t. \( ds \) is defined as the
length of the union of all validity intervals of the matches:

\[ \text{supp}(p) = \text{length} \left( \bigcup_{k=1}^{\|M\|} v_k \right) \]

This support definition computes the length of intervals
where at least one match for a pattern can be found for
a given window size. The frequency is the probability to
find a match of a pattern at a random window position for a
given dynamic scene and window size (cf. [Höppner, 2003]).

If the support value is divided by the sequence length of
the dynamic scene plus the two times the window size mi-

Fig. 1 illustrates the matching of a pattern and the cov-
ered support interval by this match ((21, 47)). The pattern
in this examples matches the first time at window start po-

cipdate in DSS.

Table 1: Composition table for the temporal relations

<table>
<thead>
<tr>
<th>B ( \tau_2 ) C</th>
<th>( \tau_1 )</th>
<th>( \tau_2 )</th>
<th>( \tau_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( &lt; )</td>
<td>( &lt; )</td>
<td>( &lt; )</td>
<td>( &gt; )</td>
</tr>
<tr>
<td>( &lt;c )</td>
<td>( &lt;c )</td>
<td>( &lt;c )</td>
<td>( &gt;c )</td>
</tr>
<tr>
<td>( = )</td>
<td>( &lt;c )</td>
<td>( &lt;c )</td>
<td>( &gt;c )</td>
</tr>
<tr>
<td>( &gt; )</td>
<td>( &lt;c )</td>
<td>( &lt;c )</td>
<td>( &gt;c )</td>
</tr>
</tbody>
</table>

In order to satisfy the temporal restriction \( tr \) it must hold
that \( \forall r, s : ir((s_r, e_r), (s_s, e_s)) \in TR[r, s] \) with \( 1 \leq r < n \)
and \( r < s \leq n \).

As the frequency of a pattern is directly related to its sup-
port we first introduce how the support is computed in our
case. In the task of frequent pattern discovery in logic, [De-
haspe, 1998] introduced an extra key parameter in order to
determine what is counted. Entities are uniquely identified
by each binding of the variables in key [Dehaspe, 1998, p.
34]. A disadvantage of this support definition is that the key
parameter must be part of each pattern in order to get a sup-
port > 0. Thus, it is not possible to compare two different
patterns if they do not share this key parameter.

We decided to use the observation time semantic for sup-
port computation as stated by Höppner. Here, the support
is defined as “the total time in which (one or more) instances
of \( P \) can be observed in the sliding window” [Höppner,
2003, p. 52]. The advantages of using observation time as
support are the clear semantics and the better efficiency
as not all matches have to be collected or maybe even fur-
ther processed. The monotonicity property for this support
definition holds and the support intervals of previous steps
(i.e., of more general patterns) can be reused in order to
restrict the search to parts of the temporal sequence in the
subsequent levels.

Definition 3.14 (Support) Let \( p \) be a temporal pattern, \( ds \)
the dynamic scene, and \( M \) the set of matches. The validity
interval of a single match \( m_i \in M \) is defined as \( v_i = [s_{max} - w + 1, e_{min} + w] \) with \( s_{max} \),
being the maximal start time and \( e_{min} \), the minimal end time of all predicate
instances in \( m_i \). The support of \( p \) w.r.t. \( ds \) is defined as the
length of the union of all validity intervals of the matches:

\[ \text{supp}(p) = \text{length} \left( \bigcup_{k=1}^{\|M\|} v_k \right) \]

This support definition computes the length of intervals
where at least one match for a pattern can be found for
a given window size. The frequency is the probability to
find a match of a pattern at a random window position for a
given dynamic scene and window size (cf. [Höppner, 2003]).

If the support value is divided by the sequence length of
the dynamic scene plus the two times the window size mi-

Fig. 1 illustrates the matching of a pattern and the cov-
ered support interval by this match ((21, 47)). The pattern
in this examples matches the first time at window start po-

cipdate in DSS.


Algorithm 1 MiTemP-main (Pattern Generation)

Input: \(ds = \{p, c, i, \delta, \Delta, \mathcal{O}, \mathcal{S}, \mathcal{I}, \mathcal{D}, \mathcal{S}_D, \mathcal{W}_D\}\), win\(_{size}\), size\(_{min}\), size\(_{max}\), minfreq\(*\)\ dynamic scene, window size, minimal and maximal pattern size, minimal frequency \(*\)

Output: All frequent patterns \(P_{freq}\) with size, \(size_{min} \leq size \leq size_{max}\)

1: Init \(P_{freq} = \emptyset\), \(i = 1\)
2: \(C_i \leftarrow \text{create_single_predicate_patterns}() /\ast\) Create one candidate for each predicate definition \(*\)
3: while \(C_i \neq \emptyset\) do
4: \(\text{support}(\mathcal{C}_i) = \text{MiTemP-support}(ds, \text{win}_{size}, C_i)\)
5: \(L_i = \{C_j \in \text{support}(C_i) : \text{maxfreq} \}\)
6: \(P_{freq} = P_{freq} \cup \{l \in L_i : \text{size} \leq \text{size}_{l_{max}} \land \text{complete temporal restriction}(l)\}\)
7: \(i \leftarrow i + 1\)
8: \(C_i = \text{MiTemP-gen-lengthening}(L_{k-1}) \cup \text{MiTemP-gen-temp-refinement}(L_{k-1}) \cup \text{MiTemP-gen-unification}(L_{k-1}) \cup \text{MiTemP-gen-class-refinement}(L_{k-1}) \cup \text{MiTemP-gen-instantiation}(L_{k-1})\)
9: end while

The goal of this work is to identify all frequent temporal patterns from a dynamic scene. In order to restrict the search space we introduce upper and lower limits for the number of predicates, i.e., for the minimal and maximal size of the conjunctive pattern, and force the temporal restriction to be completely constrained, i.e., all temporal relation sets must consist of exactly one element.

If a pattern is frequent and it satisfies these conditions we refer to it as a relevant frequent pattern. The set of these patterns forms the language \(L_{\text{MiTemP}} = \{t_{p} | t_{p} = (\mathcal{C}_p, T_{R}, cr) \land \text{freq}(t_{p}) \geq \text{minfreq} \land \text{size}_{min} \leq |\mathcal{C}_p| \leq \text{size}_{max} \land \forall i, j : |T_{R}[i,j]| = 1 \land 1 \leq i < |\mathcal{C}_p| \land i < j \leq |\mathcal{C}_p| \cup \epsilon \text{ with } |\mathcal{C}_p| > 1\}\. The most general empty pattern is denoted by \(\epsilon\).

4 MiTemP: Mining Temporal Patterns

This section introduces the MiTemP (Mining Temporal Patterns) algorithms. All algorithms are shown in pseudo code while the implementation has been realized with XSB Prolog [Sagonas et al., 2006]. The main loop for the levelwise refinement is shown in Algorithm 1 (MiTemP-main).

As mentioned above temporal patterns consist of different components like the conjunctive pattern, temporal restrictions, variable unification, class restrictions, and instantiations. For each of these components refinement operators exist that specialize a given pattern. In order to set up an optimal refinement operator which creates every pattern only once a status about the executed refinements is affixed to each pattern as it is also done by [Lee, 2006].

The status keeps track of how many refinements of each type have been performed and which position (predicate pair for temporal refinement, variable position for unification, class restriction, or instantiation) has been processed at last. A status \(\text{status}(p) = (l, t, i_{last}, u, u_{last}, c, c_{last}, k, i_{last})\) where \(l, t, u, c, i\) are the number of refinement operations of the different types, namely lengthening, temporal refinement, unification, class refinement, and instantiation; \(i_{last}, u_{last}, c_{last}, k, i_{last}\) refer to the last position where a temporal refinement, unification, class refinement, or instantiation has been performed. Similar to [Lee, 2006] we define the following refinement operations:

- **Lengthening \(\rho_L(p)\):** Adding an atom to the end of a conjunctive pattern
- **Temporal refinement \(\rho_T(p)\):** Adding a temporal constraint between two predicates in the conjunctive pattern at the leftmost position after the previous temporal refinement
- **Unification \(\rho_U(p)\):** Unifying a variable \(v_j\) with a previous one \(v_i (i < j)\) in the conjunctive pattern where no variable \(v_k\) with \(k > j\) has been unified before
- **Class refinement \(\rho_C(p)\):** Specializing a class \(c_j\) in the class restriction for the variables of the conjunctive pattern where no class restriction has been performed to any \(c_j\) with \(j > i\)
- **Instantiation \(\rho_I(p)\):** Instantiating a variable \(v_j\) of the conjunctive pattern with an instance \(o \in \mathcal{O}\) where no variable \(v_j\) has been instantiated with \(j > i\). It must also hold, that no variable \(v_k\) with \(k \neq i\) has been instantiated to \(o\).

The refinement operator is defined as the union of these operators: \(\rho(p) = \rho_L(p) \cup \rho_T(p) \cup \rho_U(p) \cup \rho_C(p) \cup \rho_I(p)\). While [Lee, 2006] also defines a “deepening” operator (for replacing a variable by a functor) which is omitted here we introduce the class refinement operator which exploits the class hierarchy of the dynamic scene schema. Another difference is that the temporal refinement here adds arbitrary temporal relations between time intervals while the “promotion” operator of Lee replaces the before relation between two events by a directly before relation.

Certain rules for each refinement coordinate when which refinement step is allowed. The lengthening operation is just allowed as long as no other refinement type has been applied and the maximal size of the conjunctive pattern is not exceeded. In order to perform a temporal refinement the minimum pattern size must be met, and no other refinement (except lengthening) must have been applied to the pattern. As we are looking for temporally completely constrained patterns the temporal refinement is only allowed to refine the next not yet processed predicate pair in the sequence. In the refinement step itself one of the possible temporal relations \(tr \in T_{R}[i,j]\) is selected. After this step the composition table (Table 1) is used to further restrict the following temporal relations. Only those patterns where all predicate pairs are restricted to one temporal relation are further processed by other refinement types.

Algorithm 2 MiTemP-gen-lengthening (Lengthening Candidate Generation)

Input: \(L_{i-1}\) / Frequent patterns of the previous step \(*\)

Output: New candidate patterns \(C_i\)

1: \(F_{i-1} = \{l \in L_{i-1} | \text{lengthening} \text{ allowed}(l)\}\)
2: for \((p_i \in F_{i-1})\) do
3: for \((p_j \in F_{i-1} \land j > i)\) do
4: if \(p_i = (ap_{i,1} \land ap_{i,2} \land \ldots \land ap_{i,i-2} \land ap_{i,i-1}) \land p_j = (ap_{i,1} \land ap_{i,2} \land \ldots \land ap_{i,i-2} \land ap_{i,j-1})\) then
5: \(p_{new1} = (ap_{i,1} \land ap_{i,2} \land \ldots \land ap_{i,i-2} \land ap_{i,j-1} \land ap_{i,j})\)
6: \(p_{new2} = (ap_{i,1} \land ap_{i,2} \land \ldots \land ap_{i,i-2} \land ap_{i,j-1})\)
7: is Add if all subsets are frequent (prune step) \(*\)
8: if \(\neg p_{sub} \subseteq p_{new1} : p_{sub} \in L_{i-1}\) then
9: \(C_i \leftarrow C_i \cup p_{new1}\)
10: end if
11: if \(\neg p_{sub} \subseteq p_{new2} : p_{sub} \in L_{i-1}\) then
12: \(C_i \leftarrow C_i \cup p_{new2}\)
13: end if
14: end if
15: end for
16: end for
variable, at \( \rho_{c}^{-1}(p) \) the most right restricted class is replaced by its single super class, at \( \rho_{T}^{-1}(p) \) all occurrences of the rightmost instantiated variable are replaced by a new variable. Assuming there exist two different paths (i.e., redundancy is given) from the most general empty pattern to a pattern \( p \in \mathcal{L}_{M\text{TemP}} \) and \( \mathcal{P}_0 = \epsilon, p_1, \ldots, p_n = p \) and \( s_0 = \epsilon, s_1, \ldots, s_n = p \) with \( r_{i+1} = \rho_i \) and \( s_{i+1} = \rho_i(s_i) \). If the inverse refinement operator is applied to both \( r_m \) and \( s_n \) the resulting sequences must be identical due to the property of the inverse refinement operator with \( r_m = s_n, r_{m-1} = s_{n-1}, \ldots, r_1 = s_1, r_0 = s_0 = \epsilon \) and \( m = n \) which contradicts the assumption of the two different paths. Thus, it follows that \( \rho \) is non-redundant.

In order to show completeness it is necessary to prove that for each pattern \( p \in \mathcal{L}_{M\text{TemP}} \) a path \( p_0 = \epsilon, p_1, \ldots, p_n = p \) exists. Here, again the inverse refinement operator and the status can be used. Let \( p \) be any pattern in \( \mathcal{L}_{M\text{TemP}} \). This path has the status \( \text{status}(p) \) with the refinement level \( n = |\text{status}(p)| \). If we get \( p_{n-1} = \rho^{-1}(p_n) \) then \( p_n \in \rho(p_{n-1}) \) and \( |\text{status}(p_{n-1})| + 1 = |\text{status}(p)| \). Referring to Lee we can find any \( p_i \) with \( 0 \leq i \leq n - 1 \) by applying \( p_{i+1} = \rho^{-1}(p_i) \) and we know that \( |\text{status}(p_i)| = i \) and \( |\text{status}(p_0)| = 0 \). By definition, the empty pattern is the only one with a refinement level of 0. Thus, we have found a sequence \( p_0 = \epsilon, p_1, \ldots, p_n = p \) with \( p_{i+1} = \rho(p_i) \).

The candidate generation algorithm for lengthening differs from the other refinements as it is not applied to each pattern separately but to the set of frequent patterns of the previous step. The algorithm (Algorithm 2) is similar to \textit{apriori-gen} [Agrawal and Srikant, 1994]. Starting from single predicate patterns in each following step patterns with the same \( n - 1 \) prefix are combined in order to create new pattern candidates (cf. [Lee, 2006]). The difference here is that the “items” in the list are actually predicates which can appear multiple times in a conjunctive pattern. As the predicate order is also relevant for distinguishing the patterns no alphanumeric order can be used to just create one new candidate of two previous frequent patterns with identical prefix. Here, two patterns must be generated.

Algorithm 3 shows the support computation procedure. Input to the algorithm are the dynamic scene, the size of the sliding window, and the list of patterns to check. As long as the latest end time is not reached a window is moved over the sequence. At each window position just the “visible” predicates identified by the window position are taken into account for pattern matching. This has the advantage that during pattern matching many assignments do not need to be checked as they are out of range of the sliding window anyway. If a match is found for a pattern at the current window position the support interval list is extended by the support interval of the match and the next position to check
is assigned. If the match is valid beyond the window border some pattern matching steps can be omitted before the pattern has to be checked again. Finally, the window position is moved to the next position.

5 Learning Temporal Patterns with WARMR

As the temporal validity intervals of predicates can be seen as just another dimension of relations it should be possible to transfer the learning problem to relational association rule mining. Intuitively, it seems to be unhandy but feasible to add information about start and end time to every predicate. We developed a converter which automatically transfers a MiTemP input file to ACE input files. ACE is a data mining system which provides a number of different relational data mining algorithms including WARMR [Blockeel et al., 2002; 2006]. Different problems had to be solved in order to set up WARMR to mine the same frequent patterns (with identical support calculation) as it is done by MiTemP. Due to space restrictions it is not possible to go into detail how the WARMR input is generated. A separate report covering these details is currently written.

The transformation of the class hierarchy and corresponding instances is straight forward. The directSubClassOf and directInstanceOf relations can be kept and put to ACE’s knowledge base file. The transitive clauses for querying instances of classes and subclasses of a class can also be left unchanged and put into the background knowledge file. The holds predicates representing the validity intervals of relations are now represented by relations with an additional argument which stands for the time interval. The predicate instance holds\( (\text{pass}(p8, p7), \langle 32, 45 \rangle) \) is converted to pass\( (1, p8, p7, i(32, 45)) \) where the first argument is a unique predicate ID.

For setting up the learning bias in WARMR it is possible to define rmode statements. These statements define how a query can be extended during the generation of new query candidates. It is also possible to define constraints which must be satisfied in order to add an atom to the query. More details about the rmodes can be found, for instance, in Dehaspe’s doctoral thesis and the ACE user’s manual [Dehaspe, 1998; Blockeel et al., 2006].

For each given MiTemP refinement as described in section 4 rmodes must be defined. For lengthening a rmode must be defined for each predicate definition. In order to avoid the same predicate instance being used more than once it must be guaranteed that the predicate ID variable differs from all other predicate ID variables of this query.

Temporal relations between intervals are represented by clauses which check if the temporal relation actually holds for the interval pair, i.e., for each temporal relation a clause exists and a rmode is created. In order to refine a pattern by adding a temporal constraint one of the temporal clauses is added to the query by relating two intervals of existing predicates of the query to each other.

Unification is handled by a special unification clause which unifies two existing variables in the previous query. The rmode declarations of ACE also provide means to define rmodes which do not introduce a new variable in the new atom but reuse an existing one. However, our intended solution should also cover the instantiation of variables (i.e., using constants). Setting up rmodes for all cases (unification, constants, and new variables) and their combinations in predicates with an arbitrary (potentially large) number of arguments would have lead to a huge number of rmodes for the predicates. Thus, if a new predicate is added to the query all arguments are new variables in the beginning. These can be unified with another variable or can be bound to a constant in further refinement steps.

For instantiation a rmode definition allows a variable to be unified with an instance. The set of instance candidates depends on the predicate where the variable occurs. Only those instances are taken into account which appear at least in one of the predicates at the variable’s position in the dynamic scene, i.e., no “impossible” query will be generated.

Class refinement is performed by adding instanceOf predicates, constraining a variable to a certain class (or one of its sub classes). A constraint definition makes sure that for each variable just one instanceOf predicate will be added. Additional constraints ensure that a variable will be used just for instantiation or class refinement and that unified variables are not refined at all.

Setting up WARMR for computing the support as intended was a little bit trickier. WARMR needs a counting attribute which is used for support computation, i.e., the number of different values of this attribute where a query matches determines the support of the query. In our case the support is defined to be the number of temporal positions where within a sliding window a pattern holds. In order to let WARMR compute the intended support a predicate currentIndex has been introduced and used as counting attribute. For each existing temporal position a predicate is created in the knowledge base file. In combination with another predicate representing the window position \((\text{inWindowPos})\) for each temporal position it can be checked if a pattern holds.

Some tricks have made it possible to use WARMR as in-
tended for mining temporal patterns. However, we had to accept some compromises in the solution. Computing the support is a little bit more inefficient as necessary as there is no way to use a real sliding window which covers an extended interval. In the current solution each time position has to be checked on its own (even if it can theoretically be known that the pattern holds at the current position due to the sliding window of an earlier position) and also predicate instances outside of scope of the window might be checked.

Another problem is that redundant patterns are generated by WARMR. To the best of our knowledge it is not possible to avoid that for a unification two patterns are generated \((A = B \land B = A)\). Furthermore, different representations can be generated for the same pattern if additional restrictions could be derived from the pattern (e.g., temporal relations using the composition table, class restrictions which should be specialized due to unification of variables as it is done by MiTemP).

In the case of MiTemP we required each pattern to be completely constraint w.r.t. temporal relations, i.e., that for each predicate pair exactly one temporal relation should be assigned. To the best of our knowledge it is not possible to define a constraint in ACE which guarantees to just create patterns which satisfy this property. This leads to the generation of some patterns which are out of scope of MiTemP.

Even though WARMR might have some drawbacks for our temporal pattern mining task it should be stated clearly that WARMR is not a special solution for mining temporal patterns from such an interval representation but a generic system for mining frequent queries which also can be used to mine queries representing temporal patterns.

6 Evaluation

The experiments with WARMR and MiTemP have different goals. First of all, it is a proof of concept that both approaches can be used to mine frequent temporal patterns. In order to find out if both approaches lead to the same support values the frequencies of all common patterns are compared. Furthermore, it is expected that constraining the search space at the refinement steps of the algorithm reduce the number of generated patterns a lot.

For the evaluation a simple soccer scenario has been used (Fig. 2). Different objects in the dynamic scene are objects q6 - q9 of class team1 and objects p6 - p9 of class team2. Relations between these objects can be uncovered, closerToGoal, and pass. Fig. 3 shows the temporal validity intervals of the relations between the objects (time proceeding from left to right).

In different experiments the maximal refinement level of WARMR and MiTemP has been altered from five to eight. As WARMR could not create any complete pattern as defined above with a maximal refinement level below five these settings have been left out here. Table 2 summarizes the results of the test runs. Besides the number of created, redundant, and frequent patterns for both approaches it is also shown how many patterns have just been found by one approach up to this level and how many common patterns have been found by both approaches. The last two columns show the coverage, i.e., how many patterns of the other approach are covered at the current level. Fig. 4 shows a graph comparing the number of generated patterns at the different maximal refinement levels. Fig. 5 compares the number of mined frequent patterns and the number of redundant patterns of both approaches for the different levels.

All common patterns of WARMR and MiTemP get identical frequency values assigned. Fig. 6 shows example outputs of the same pattern by WARMR and MiTemP. While WARMR creates a number of redundant patterns (growing with an increasing maximal refinement level) the refinement operators in MiTemP are optimal as no redundant pattern was created (dotted lines in Fig. 5). MiTemP identifies much more frequent patterns at the different maximal refinement levels and creates less patterns at maximal refinement levels seven and eight– at levels five and six MiTemP creates more patterns. While the fraction of relevant frequent patterns to created patterns is quite low with WARMR (at level eight it is \(\frac{6530}{10000} = 65.3\%\)) almost every second pattern created by MiTemP is a relevant frequent one (at level eight: \(\frac{17940}{39889} = 44.9\%\)).

In the eighth level some patterns which have been mined by WARMR have not yet been found by MiTemP at this level. An inspection of the patterns has shown that these are patterns with many instantiations. Due to the refinement structure in MiTemP a variable is not instantiated before the class refinement restricts the variable to a leaf class (i.e., having no sub classes). In the WARMR solution in-
stantiation can be performed directly to a variable, i.e., the intermediate class refinement steps (specializing variables to team1 or team2) are not needed and thus, some patterns can be created at an earlier refinement level.

7 Conclusion

In this paper we have presented an approach to temporal pattern mining which mines frequent patterns from time interval-based relational representations of dynamic scenes. An Apriori-like algorithm has been introduced which performs a top-down search of the pattern space without multiple generation of patterns. The use of reasoning techniques creates the most specialized representation after refinement or identifies inconsistencies in patterns. This avoids the creation of “impossible” patterns which cannot be frequent as well as reduces the number of specialization steps which are implicit in the pattern already. Here, a composition table is used for identifying possible temporal refinements, and class information of variables is used to find the most special class of a variable by taking into account predicate definitions, variable unifications, and instantiations. An implementation of the well-known WARMR algorithm has been used to create another solution for the mining problem. The experiments have shown the advantage of avoiding the creation of impossible or redundant patterns in MiTemp. While less patterns had to be explored at refinement levels seven and eight much more frequent patterns have been found by MiTemp already. This can be particularly of importance if large sequences have to be processed, i.e., if support computation is costly.

Acknowledgment

We would like to thank Frank Höppner at the Fachhochschule Braunschweig/Wolfenbüttel for helpful discussions on support computations in temporal pattern mining. We also want to express our gratitude to the members of the DTAI research group of the KU Leuven, Belgium, for providing the ACE system including WARMR [Blockeel et al., 2002]. Particularly, we would like to thank Jan Struyf for his great support with ACE/WARMR.

References


