An Interactive Algorithm for the Complete Discovery of Chronicles

Research Report

Damien Cram, Amélie Cordier and Alain Mille
Université de Lyon, CNRS
Université Lyon 1,LIRIS, UMR5205, F-69622, France

April 22, 2009

Abstract

Chronicles are temporal patterns with numerical constraints extracted from a sequence of events. The representation formalism of chronicles is very expressive compared to classical sequential patterns like serial and parallel episodes. Due to this expressiveness, any complete discovery process of chronicles has a very high time complexity. In this article, we propose an interactive approach and an algorithm for a complete discovery of chronicles. The algorithm has been designed in order to provide heuristic-search facilities and to allow users to define their own constraints. The interactive discovery approach combines these two features thus taking into account the user's knowledge when exploring the space of candidate chronicles. Thanks to this approach, the most interesting chronicles from the user's point of view are expected to be extracted first and in an acceptable time thus avoiding the user to wait for the complete set of frequent chronicles to be extracted. [1]

[1]This research is supported by the french National Research Agency (ANR) as a contribution to the project PROCOGEC (see www.procogec.com), in collaboration with members of the project SharedLife from DFKI (www.dfki.de/sharedlife).

1 Introduction

Knowledge discovery from interaction traces requires the ability to find temporal patterns that are relevant to a given research question. This ability relies both on appropriate mining algorithms and on the user's initial knowledge about the behavior that is represented by these sequences [1]. Blind search based on classical mining criteria produces a large number of candidate patterns for representing possibly relevant pieces of knowledge. Selecting relevant patterns among the results of such a mining remains tedious and constitutes a strong impediment of the method. This led researchers to favor non-complete approaches in order to reduce the complexity and to control better the mining process. New approaches are directed towards the development of interactive visualization tools providing users with synthetic and efficient visualization features and making them able to integrate their own knowledge on the algorithm in order to guide the discovery process [2]. The approach described hereafter belongs to this field of research. It is a complete algorithm that provides us with a mean of performing truly exploratory yet constraint mining, thus avoiding to exclude possibly relevant patterns. As the complexity of such an algorithm is large, it is necessary to provide it with the ability to take into account constraints related to the knowledge of the observer. This paper presents a complete variant of an existing algorithm of discovery of chronicles [3] in temporal sequences of events that takes into account constraints and that have an anytime behavior. Chronicles are temporal patterns over sequences of events like serial and parallel episodes [4], but with additional numerical temporal constraints on events. These numerical temporal constraints are needed to represent better typical situations in interaction traces. For example, two chronicles having the same events but different temporal constraints can represent the same user task in interaction traces, but the one whose constraints are shorter could represent the execution of this task by an expert of the traced system and the one with large constraints could represent the task executed by a novice. The complete approach is formalized (cf. sections 2, 3 and 5) and demonstrated (cf. section 4). The complexity issue is described and the way we tackle it is discussed (cf. section 6). We illustrate the potential of the approach by showing how an interface may assist the user in his task in a concrete application domain: an experimental instrumented kitchen.

2 Definitions

2.1 Chronicle

We define a *sequence of events* as an ordered set, denoted $\mathcal{S} = \langle (e_1, t_1) \dots (e_l, t_l) \rangle$. Each pair (e_i, t_i) is an *event*, where e_i is an *event type* and t_i and integer called the *event date*. Each *event type* e_i is an element of the finite and totally ordered set of event types $(\mathbb{E}, \leq_{\mathbb{E}})$. The temporal patterns we search for are *chronicles* [3]. To define what a *chronicle* is, we need to first introduce the concept of *temporal constraint*. A *temporal constraint* is a quadruplet $(\varepsilon, \varepsilon', I^-, I^+)$, denoted $\varepsilon[I^-, I^+]\varepsilon'$, where $(\varepsilon, \varepsilon') \in \mathbb{E}^2$ and I^- and I^+ are two integers such that $I^- \leq I^+$. Two events (e_1, t_1) and (e_2, t_2) are said to satisfy the constraint $\varepsilon[I^-, I^+]\varepsilon'$ if $e_1 = \varepsilon$ and $e_2 = \varepsilon'$ and $t_2 - t_1 \in [I^-, I^+]$, or if $e_1 = \varepsilon'$ and $e_2 = \varepsilon$ and $t_2 - t_1 \in [-I^+, -I^-]$. A *chronicle* is a pair $\mathcal{C} = (\mathcal{E}, \mathcal{T})$, such that:

- $\mathcal{E} = \varepsilon_1 \dots \varepsilon_n$, where $\forall i, \varepsilon_i \in \mathbb{E}$ and $\forall i < j, \varepsilon_i \leq_{\mathbb{E}} \varepsilon_j$; ($n$ is the *size* of $\mathcal{C}$)
- $\mathcal{T} = \{\tau_{ij}\}_{1 \leq i < j \leq |\mathcal{E}|}$ is a set of temporal constraints on $\mathcal{E}$ such that $\forall i < j, \tau_{ij} = \varepsilon_i[\tau_{ij}^-, \tau_{ij}^+]\varepsilon_j$. ($\mathcal{E}$ is the *episode* of $\mathcal{C}$)

An *occurrence* of $\mathcal{C}$ is a list of events in $\mathcal{S}$, denoted $\langle (\varepsilon_1, t_1) \dots (\varepsilon_n, t_n) \rangle\}$, that satisfies all temporal constraints defined in $\mathcal{C}$, i.e. $\forall i < j, t_j - t_i \in [\tau_{ij}^-, \tau_{ij}^+]$.

Given $\mathbb{E} = \{A, B, C\}$ and $\mathcal{S}_0 = \langle (A,1)(C,2)(B,4)(A,5)(C,5)(B,6) \rangle$, occurrences of the chronicle $\mathcal{C}_1 = (ABC, \{A[1,3]B, A[1,1]C, B[-2,-1]C\})$ are $\langle (A,1)(C,2)(B,4) \rangle$ and $\langle (A,5)(C,5)(B,6) \rangle$ (cf. first part of figure 1).

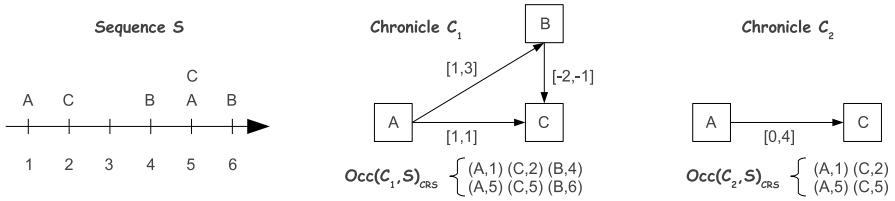


Figure 1: An event sequence $\mathcal{S}_0$ and two chronicles $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $\mathcal{C}_1 \preceq \mathcal{C}_2$.

A chronicle $\mathcal{C} = (\mathcal{E}, \mathcal{T})$ is *stricter than* a chronicle $\mathcal{C}' = (\mathcal{E}', \mathcal{T}')$, or $\mathcal{C}'$ is *more general than* $\mathcal{C}$, if 1) the episode $\mathcal{E}'$ is a subepisode of $\mathcal{E}$, i.e. if

there is an increasing function h, called the *inclusion function*, such that $\mathcal{E}' = \varepsilon'_1 \dots \varepsilon'_{n'} = \varepsilon_{h(1)} \dots \varepsilon_{h(n')}$, and if 2) $\forall i < j, [\tau^-_{ij}, \tau^+_{ij}] \preceq [\tau^-_{h(i)h(j)}, \tau^+_{h(i)h(j)}]$. The *stricter than* relation between two chronicles is denoted $\preceq$. "$\preceq$" is a partial relation of order in the set of all chronicles. For example on figure 1, $\mathcal{C}_1 \preceq \mathcal{C}_2$. We also define the relation *strictly stricter than*, denoted "$\prec$", defined by $\mathcal{C} \prec \mathcal{C}' \Leftrightarrow \mathcal{C} \preceq \mathcal{C}'$ *and* $\mathcal{C} \neq \mathcal{C}'$. More accurately, $\mathcal{C}_1 \prec \mathcal{C}_2$ since $\mathcal{C}_1 \neq \mathcal{C}_2$.

2.2 Frequency of a chronicle

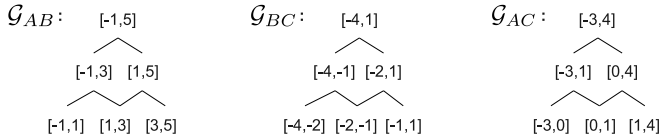
Given an event sequence and a chronicle $\mathcal{C}$, we define the frequency of $\mathcal{C}$ in $\mathcal{S}$ as the number of chronicles recognized by the algorithm `CRS` (Chronicle Recognition System), the chronicle recognition algorithm presented in [5] and improved in [6]. $\mathcal{C}$ is said to be frequent if its frequency $f_{CRS}(\mathcal{C})$ is greater than a threshold frequency value f_{th}. It has been proven in [3] that `CRS` is monotonic for $\preceq$. This property is very convenient when discovering chronicles in a *generate-count* way like we do in section 3.

2.3 Constraint-database

The definition of the notion of *constraint-database*, needed in the formulation of the complete discovery problem, requires to define first the notion of *constraint-graph*. Given $(x, y) \in \mathbb{E}^2$, $\mathcal{G}_{xy}$ is said to be a *constraint-graph* on (x, y) if $\mathcal{G}_{xy}$ is an acyclic oriented graph such that:

1. each node is a temporal constraint on $\tau = \varepsilon[\tau^-, \tau^+]\varepsilon'$ such that $\varepsilon = x$ and $\varepsilon' = y$ (each node in $\mathcal{G}_{xy}$ is a temporal constraint on (x, y)),
2. for each pair (τ, τ') of nodes in $\mathcal{G}_{xy}$, $\tau = x[\tau^-, \tau^+]y$ and $\tau' = x[\tau'^-, \tau'^+]y$, there is an arrow from τ to τ' if and only if $\tau' \prec \tau$ and there exists no τ'' in $\mathcal{G}_{xy}$ such that $\tau' \prec \tau'' \prec \tau$.

A *constraint-database* $\mathcal{D}$ is a set of constraint-graphs $\mathcal{G}_{\varepsilon_i\varepsilon_j}$, where for all (i, j) $(\varepsilon_i, \varepsilon_j) \in \mathbb{E}^2$ and $\varepsilon_i \leq_{\mathbb{E}} \varepsilon_j$. By constraining $\varepsilon_i \leq_{\mathbb{E}} \varepsilon_j$ we ensure that given two event types ε_i and ε_j there will never be two graphs $\mathcal{G}_{\varepsilon_i\varepsilon_j}$ and $\mathcal{G}_{\varepsilon_j\varepsilon_i}$ in $\mathcal{D}$. In order to simplify the mining algorithm presented in section 3, we also assume that each $\mathcal{G}_{ij}$ in $\mathcal{D}$ contains exactly one constraint, denoted $\mathcal{G}^\top_{ij}$ and called the *root constraint*, such that $\forall \tau \in \mathcal{G}_{ij}\ \tau \preceq \mathcal{G}^\top_{ij}$. Figure 2 shows the example of the constraint-database $\mathcal{D}_0$.



$$\begin{aligned}
\mathcal{D}_0 &= \{((A,B),\mathcal{G}_{AB}),((B,C),\mathcal{G}_{BC}),((A,C),\mathcal{G}_{AC})\} \\
\mathcal{G}_{AB} &= \{A[-1,5]B, A[-1,3]B, A[1,5]B, A[-1,1]B, A[1,3]B, A[3,5]B\} \\
\mathcal{G}_{AB}^{\top} &= A[-1,5]B \\
\mathcal{D}_0^{\top} &= \{A[-1,5]B, B[-4,1]C, A[-3,4]C\}
\end{aligned}$$

Figure 2: $\mathcal{D}_0$: an example of a constraint-database.

2.4 $\mathcal{D}$-based chronicle and the *straight stricter than* relation

The complete discovery problem can be stated on $\mathcal{D}$-chronicles only. A $\mathcal{D}$-chronicle is a chronicle whose constraints are taken from the contraint-database $\mathcal{D}$. Formally:

$$(\mathcal{E},\mathcal{T}) \text{ is a } \mathcal{D}\text{-chronicle} \Leftrightarrow \forall \tau_{ij} \in \mathcal{T},\ \tau_{ij} \in \mathcal{D}$$

For instance on figure 1, $\mathcal{C}_2$ is a $\mathcal{D}_0$-chronicle while $\mathcal{C}_1$ is not, since $A[1,3]B \notin \mathcal{G}_{AB}$ (actually, $B[-2,-1]C \notin \mathcal{G}_{BC}$ and $A[1,1]C \notin \mathcal{G}_{AC}$ also).

Given a constraint-database $\mathcal{D}$, a chronicle $\mathcal{C}'$ is said to be *straight stricter than* $\mathcal{C}$ against $\mathcal{D}$ when:

1. $\mathcal{C}$ et $\mathcal{C}'$ are $\mathcal{D}$-chronicles,
2. and $\mathcal{C}' \prec \mathcal{C}$ ($\mathcal{C}'$ *strictly* stricter than $\mathcal{C}$),
3. and there is no $\mathcal{D}$-chronicles $\mathcal{C}''$ such that $\mathcal{C}' \prec \mathcal{C}'' \prec \mathcal{C}$ (*strictly*).

For instance on figure 3 all chronicles are $\mathcal{D}_0$-chronicles. On that figure we observe that $\mathcal{C}_7 \prec \mathcal{C}_3$. However, $\mathcal{C}_7$ is not *straight stricter than* $\mathcal{C}_3$ since $\mathcal{C}_4$ is a $\mathcal{D}_0$-chronicle and $\mathcal{C}_7 \prec \mathcal{C}_4 \prec \mathcal{C}_3$. Nevertheless, $\mathcal{C}_4$ is *straight stricter than* $\mathcal{C}_3$ and $\mathcal{C}_7$ is *straight stricter than* $\mathcal{C}_4$.

3 Complete solving algorithm

3.1 Problem statement

The complete discovery of chronicles can be stated as follows:

Given an event sequence $\mathcal{S}$, a minimum number of occurrences f_{th} and the constraint-database $\mathcal{D}$ return by algorithm `CCDC` (alg. 3) when applied on $\mathcal{S}$ and f_{th}, find all minimal frequent $\mathcal{D}$-chronicles $\mathcal{C}$ such that $f_{CRS}(\mathcal{C}) \geq f_{th}$.

The statement of the complete discovery problem is based on the constraint-database $\mathcal{D}$ that must be given as input. The chronicle discovery problem cannot be considered as *complete* as long as the $\mathcal{D}$ is not considered as *complete* neither. The issue of building $\mathcal{D}$ in a *complete* way is discussed in section 5 together with other types of constraint-databases.

The basic principle of the solving algorithm is to generate candidate $\mathcal{D}$-chronicles and to count them, in the style of *Apriori* [7]. Step by step, new candidates are generated from frequent $\mathcal{D}$-chronicles by taking one frequent $\mathcal{D}$-chronicle $\mathcal{C}$ and computing all $\mathcal{D}$-chronicle $\mathcal{C}'$ such that $\mathcal{C}'$ is *straight stricter than* $\mathcal{C}$.

3.2 Candidates generation

We can generate all successors of a $\mathcal{D}$-chronicle $(\mathcal{E}, \mathcal{T})$ by using the two following types of operations:

`add_`ε adds the event type ε to $\mathcal{E}$. This operation is a bit more tricky than it looks since the addition of the event type ε to $\mathcal{E} = \varepsilon_1 \dots \varepsilon_{n-1}$ also implies the addition of all root constraints $\mathcal{G}^{\top}_{\varepsilon_k \varepsilon}$ $(1 \leq k \leq n-1)$ to $\mathcal{T}$. If for any k there is no entry for the pair $(\varepsilon_k, \varepsilon)$ in $\mathcal{D}$, then `add_`ε cannot be applied to $(\mathcal{E}, \mathcal{T})$.

`str_`$\varepsilon_i\varepsilon_j$ takes any constraint $\tau_{ij} = \varepsilon_i[\tau^-_{ij}, \tau^+_{ij}]\varepsilon_j$ in $\mathcal{T}$ and replaces it with $\sigma_{ij} = \varepsilon_i[\sigma^-_{ij}, \sigma^+_{ij}]\varepsilon_j$, where σ_{ij} is *straight stricter than* τ_{ij} in the constraint-graph $\mathcal{G}_{\varepsilon_i\varepsilon_j}$ in $\mathcal{D}$. If $\mathcal{D}$ doesn't contain any constraint-graph on the pair $(\varepsilon_i, \varepsilon_j)$ or if $\mathcal{G}_{\varepsilon_i\varepsilon_j}$ exists in $\mathcal{D}$ but contains no constraint stricter than τ_{ij}, then `str_`$\varepsilon_i\varepsilon_j$ cannot be applied to $(\mathcal{E}, \mathcal{T})$.

For instance on figure 3, $\mathcal{C}_5$ is generated from $\mathcal{C}_3$ by applying `add_B` (the addition of the event type B to $\mathcal{C}_5$) and $\mathcal{C}_4$ is generated from $\mathcal{C}_3$ by applying `add_AC` (the strengthening of $A[-3,4]C$ into $A[-3,1]C$). In total, there are $|\mathbb{E}|$ operations of type `add`, and $\sum_{\tau_{ij} \in \mathcal{T}} n_{ij}$ operations of type `str`, where n_{ij} is the number of straight stricter constraints of τ_{ij} in $\mathcal{D}$.

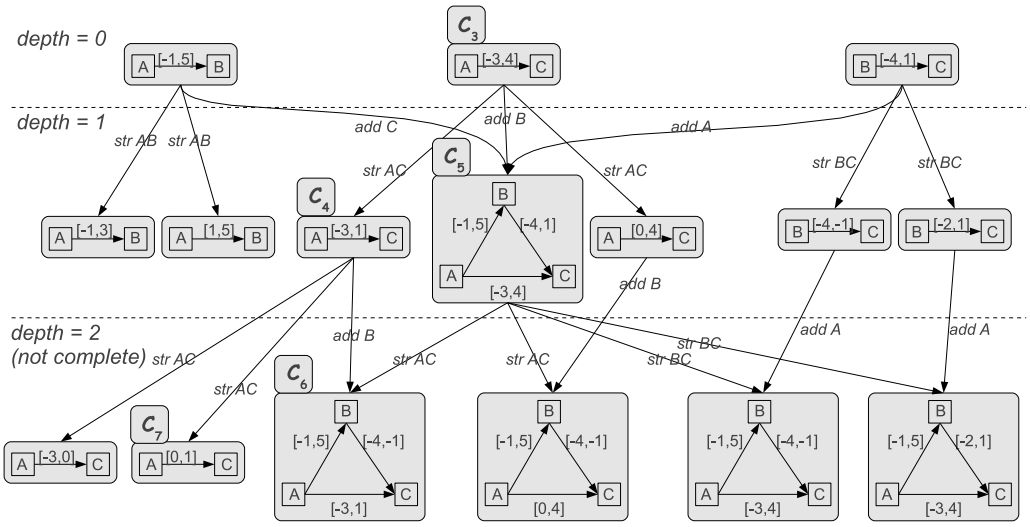


Figure 3: Successor-graph of $\mathcal{D}_0$-chronicles for the three first depth levels. All chronicles of depth 0 and 1 are on the figure; level 2 is given partially. The set of chronicles whose depth is 0 is $\mathcal{D}_0^\top$.

From this point, we consider that $\mathcal{C}$ is a *successor* of $\mathcal{C}'$ if $\mathcal{C}$ is *straight stricter than* $\mathcal{C}'$ against $\mathcal{D}$. We also define the relation *predecessor* as the inverse relation of *successor*.

3.3 Algorithm

Algorithm `HDA` applies this principle. `Open_chrs` is the set of candidate chronicles whose frequencies in $\mathcal{S}$ must be tested and `Freq` is the set to be returned. When `Open_chrs` is empty, the algorithm ends (line 23). The iteration 3-23 implements the generation-counting approach explained above, one candidate chronicle at a time. The process is initiated with all 2-chronicles that are in $\mathcal{D}^\top$ (lines 1-2).

During an iteration the following process is applied. Any chronicle $\mathcal{C}$ in the waiting candidates list is chosen and removed from `Open_chrs` (line 4). Instead of directly counting it against $\mathcal{S}$, which costs a lot of time, the frequency of $\mathcal{C}$ is first pre-tested (lines 5-8). To do that, we maintain a set `non_Freq` of chronicles that have been proved not to be frequent. If `non_Freq` contains a chronicle that is stricter than $\mathcal{C}$, then $\mathcal{C}$ can be dropped since $\mathcal{C}$

Algorithm 1 `HDA`: Heuristic Discovery Algorithm

```
Require: S ; D; f_th
Ensure: Freq
 1: Freq← ∅
 2: Open_chrs ← D^⊤
 3: repeat
 4:    C ← take_first_from(Open_chrs)
 5:    if has_more_general(non_Freq,C) then
 6:       continue {line 3}
 7:    end if
 8:    if ¬has_stricter(Freq,C) then
 9:       f_CRS(C) ←count_CRS(C, S)
10:       if f_CRS(C) ≥ f_th then
11:          add_as_minimal(C,Freq)
12:       else
13:          add_as_maximal(C,non_Freq)
14:          continue {line 3}
15:       end if
16:    end if
17:    Succ(C) ← generate_successors(C)
18:    for each C' ∈ Succ(C) do
19:       if is_acceptable(C') then
20:          Open_chrs ← Open_chrs ∪ {C'}
21:       end if
22:    end for
23: until Open_chrs = ∅
24: return Freq
```

would be *a fortiori* non-frequent, due to the monotonicity of the frequency-counting algorithm `CRS` [5, 6]. Similarly, if `Freq` contains any chronicle $\mathcal{C}'$ such that $\mathcal{C}' \preceq \mathcal{C}$, it would prove that $\mathcal{C}$ is frequent without having to count it.

Lines 9 to 15 are executed if $\mathcal{C}$ has to be counted. $\mathcal{C}$ is counted at line 9 and if $\mathcal{C}$ is counted to be non-frequent, then the method `add_as_maximal` updates `non_Freq` with $\mathcal{C}$ if needed, i.e. removes from `non_Freq` any chronicle $\mathcal{C}'$ such that $\mathcal{C}' \preceq \mathcal{C}$ and adds $\mathcal{C}$ to `non_Freq` if $\mathcal{C}$ is maximal in `non_Freq` (line 13). Similarly, if $\mathcal{C}$ is counted to be frequent, then `Freq` is updated (lines 11).

Lines 17 to 22 are executed if $\mathcal{C}$ has been proved to be frequent, whether or not it has been counted. `generate_successors` applies all applicable operations of both types to generate all successors of $\mathcal{C}$ as explained above. Then, each successor $\mathcal{C}'$ is added to the waiting candidates set `Open_chrs`, if $\mathcal{C}'$ is an acceptable candidate. Algorithm `is_acceptable` (alg. 2) details what an *acceptable* candidate is. A candidate $\mathcal{C}'$ is said to be *acceptable* when $\mathcal{C}'$ satisfies all monotone user constraints and if $\mathcal{C}'$ has not been already added once to `Open_chrs`. The procedure `is_acceptable` is the one that enables to push user-constraints in the mining process (see section 6). By checking that $\mathcal{C}'$ has never been added to `Open_chrs` we ensure that iteration 3-23 of algorithm `HDA` will never process twice the same chronicle $\mathcal{C}$ and that `HDA` will always terminate (cf. section 4.1).

Algorithm 2 `is_acceptable`

Require: $\mathcal{C}'$
Ensure: **true** if $\mathcal{C}'$ is a valid candidate to be added to `Open_chrs`, **false** otherwise.
1: **if** $\mathcal{C}'$ doesn't satisfy monotone user constraints **then**
2: **return false**
3: **end if**
4: **if** `has_already_been_processed`($\mathcal{C}'$) **then**
5: **return false**
6: **end if**
7: **return true**

4 Properties of the proposed algorithm

4.1 Termination

Let's denote n_{max} the size of the biggest chronicle in the set of all frequent minimal $\mathcal{D}$-chronicles. The method `generate_successors` generates from a chronicle of size n only chronicles of size at most $n+1$; since it also applies only on frequent chronicles, we can be sure that there will never be any candidate chronicle of size $n_{max}+2$ (since it would imply that a chronicle of size $n_{max}+1$ is frequent). Consequently, there will be in the set of candidates `Open_chrs` only chronicles of size at most $n+1$. Furthermore, there is a finite number of $\mathcal{D}$-chronicles of size at most $n_{max}+1$ and algorithm `HDA` never puts twice the same chronicle into the candidate set `Open_chrs` (line 19). Therefore `HDA` terminates.

4.2 Completeness

4.2.1 Depth of a chronicle

In a temporal constraint-graph $\mathcal{G}$ as defined in section 2.3, we can define the notion of *path* to $\tau \in \mathcal{G}$ as follows:

$$\{\tau_0, \ldots, \tau_n\} \text{ is a path to } \tau \text{ in } \mathcal{G} \Leftrightarrow \begin{cases} \forall i,\ \tau_i \in \mathcal{G} \\ \tau_0 = \mathcal{G}^\top \\ \tau_n = \tau \\ \forall i,\ \text{the pair } (\tau_i, \tau_{i+1}) \text{ is an arrow of } \mathcal{G} \end{cases}$$

Since in a constraint-database $\mathcal{D}$ constraint-graphs' arrows represent *straight stricter than* relations only, we can easily prove that for all $\mathcal{G} \in \mathcal{D}$ and for all $\tau \in \mathcal{G}$, the lengths of paths to τ in $\mathcal{G}$ are equals. Thus, we can define

the *depth* of a temporal constraint τ in a temporal constraint-graph $\mathcal{G}$ as the number of arrows that lead to τ from $\mathcal{G}^\top$.

$$depth(\tau, \mathcal{G}) = |any\ path\ to\ \tau\ from\ \mathcal{G}^\top| - 1$$

For instance:
$depth(A[1,3]B, \mathcal{G}_{AB}) = 2$
$depth(A[0,4]C, \mathcal{G}_{AC}) = 1$
$\forall \mathcal{G} \in \mathcal{D}\ depth(\mathcal{G}^\top, \mathcal{G}) = 0$

Similarly, we can define the depth of any $\mathcal{D}$-chronicle $\mathcal{C} = (\mathcal{E}, \mathcal{T})$ against $\mathcal{D}$ as follows: $depth(\mathcal{C}, \mathcal{D}) = (|\mathcal{E}| - 2) + \sum_{\tau \in \mathcal{T}} depth(\tau, \mathcal{G}_{\varepsilon\varepsilon'})$, where $(\varepsilon, \varepsilon')$ is the pair of event types of each constraint τ ($\tau = \varepsilon[\tau^-, \tau^+]\varepsilon'$). The *depth* of a $\mathcal{D}$-chronicle $\mathcal{C}$ against $\mathcal{D}$ is also the number of operations that must be applied to any τ in $\mathcal{D}^\top$ such that $\mathcal{C} \preceq \tau$ before we obtain $\mathcal{C}$. Figure 3 illustrates the notion of *depth*. On figure 3, each arrow between two chronicles stands for the application of an operation. Each arrow can be read as *is a predecessor of*.

4.2.2 Proof of completeness

The completeness of algorithm `HDA` regarding the problem stated in section 3.1 can be proved first by demonstrating that the predicate $P(n)$ defined below is true for each integer $n \geq 1$:

> $P(n)$: if $\mathcal{C}_n$ is a frequent $\mathcal{D}$-chronicle of depth n, then $\mathcal{C}_n$ has at least one predecessor $\mathcal{C}_{n-1}$ to which the method `generate_successors` has been applied.

Proof
If $\mathcal{C}$ is a frequent chronicle such that $depth(\mathcal{C}, \mathcal{D}) = 1$, all its predecessors $\mathcal{C}'$ satisfy both $depth(\mathcal{C}, \mathcal{D}) = 0$ and "$\mathcal{C}'$ is frequent" (due to monotonicity). Furthermore, algorithm `HDA` *is initiated with $\mathcal{D}^\top$, in other words with all chronicles of depth 0. Consequently, the first of the predecessors of $\mathcal{C}$ that is taken by* `take_first_from` *(line 4) will satisfy the frequency test (line 10) and line 17 will be executed. Therefore,* `generate_successors` *will be called, which proves $P(1)$.*

Assuming that $P(n)$ is true implies $P(n+1)$ is also true since if $\mathcal{C}$ is a frequent $\mathcal{D}$-chronicle of depth $n+1$, then all its predecessors $\mathcal{C}'$ will be of depth n and be also frequent, due to monotonicity. Therefore, $P(n)$ applies for each $\mathcal{C}'$, which means that `generate_successors` *will be applied at least*

once to one of the predecessors of $\mathcal{C}'$, which implies that $\mathcal{C}'$ will be added to ***Open_chrs****. Back to $\mathcal{C}$, this means that each of its predecessors will be added in* ***Open_chrs*** *and will then be processed by the iteration 3-23. Let's assume that $\mathcal{C}'_1$ is the first of these predecessors to be processed by iteration 3-23. As $\mathcal{C}'_1$ is frequent, the method* ***generate_successors*** *will be applied to $\mathcal{C}'_1$ (since line 17 to 22 are executed whenever $\mathcal{C}$ is frequent, whatever it has been counted). This proves $P(n)$.*□

Therefore, if $\mathcal{C}_n$ is a frequent minimal $\mathcal{D}$-chronicle of depth n, we can be sure that it has been added to `Open_chrs` (line 20) after having been generated by one of its predecessors $\mathcal{C}_{n-1}$. This proves that $\mathcal{C}_n$ will be taken by `take_first_from` (line 4) in a later loop of iteration 3-23. $\mathcal{C}_n$'s iteration will not stop at line 6, because there is no chronicle that is both more general than $\mathcal{C}_n$ and non-frequent since $\mathcal{C}_n$ is frequent (because f_{CRS} is monotonic). Lines 9 to 15 will be run since $\mathcal{C}_n$ is minimal, which means that there can't be any frequent chronicle stricter than $\mathcal{C}_n$ in `Freq`. Therefore, $\mathcal{C}_n$ will be counted as being frequent (line 9) and added to `Freq` (line 11). As a consequence, $\mathcal{C}_n$ will stay on in `Freq` as long as no stricter frequent chronicle is added to `Freq`, which is impossible since we assumed that $\mathcal{C}_n$ is a frequent minimal chronicle. The frequent minimal $\mathcal{D}$-chronicle $\mathcal{C}_n$ is then returned by `HDA` in `Freq`, which proves the completeness.□

4.3 Complexity

Given a set of event types $\mathcal{E} = \varepsilon_1 \dots \varepsilon_n$ of size n, there are in total $C_n^2 = n \times (n-1)/2$ pairs of event types. If we assume that all constraint-graphs in $\mathcal{D}$ are each of size p, then we can make $p^{C_n^2}$ different chronicles based on $\mathcal{E}$, just by combining temporal constraints. The total number of ordered event type sets $\mathcal{E}$ of size n is lower than $|\mathbb{E}|^n$. Thus, if we denote as $N(n)$ the total number of chronicles of size n, then we have $N(n) = O(|\mathbb{E}|^n \times p^{C_n^2})$. If we denote as n_{max} the size of the biggest chronicle in the set of all frequent minimal $\mathcal{D}$-chronicles, then algorithm `HDA` will process at most $N_{all\ together} = N(1) + \ldots + N(n_{max}+1)$ chronicles:

$$N_{all\ together} = O(|\mathbb{E}|^{n_{max}} \times p^{n_{max}^2})$$

The time complexity as estimated above looks huge, but it is just an upper bound of the total number of chronicles that are actually processed

(see section 6) when applied to concrete sequences of events. Furthermore, the "anytime-like" architecture of algorithm `HDA`, where the set `Freq` contains at any time a collection of partial solutions that can be returned to the user, enables the use of user-constraints pushing technics (through method `is_acceptable`) and heuristics (through method `take_first_from`), which both play important roles in shortening the actual time before the user is satisfied with partial solutions that are contained in `Freq` (see section 6). Also, the constraint-database can be chosen in such a way that the average number of constraints p in each contraint-graph is rather small.

5 Building the constraint-database

In this paper we claim that the discovery process is complete, contrary to the one described in [3] about which we could now say that the authors decided to keep only one constraint per graph in the constraint-database, for complexity reasons. The chronicle extraction process described above can be considered as being complete only if the constraint-database that it is based on can be considered as complete too, which is not the case when there is only one temporal constraint per graph in $\mathcal{D}$. We can think of three ways of building the constraint-database:

1. the "Duong-like" building method we have just seen [3], where each pair of event types can have only one temporal constraint;
2. the complete constraint-database building; (cf. section 5.1)
3. the constraint-database to build and discover *hybrid episodes*. (cf. section 5.2)

5.1 Building the complete constraint-database

The term "complete constraint-database" refers to a constraint-database in which every temporal constraint that is frequent in $\mathcal{S}$ would be stored. For instance, if the temporal constraint $\tau = \varepsilon[\tau^-, \tau^+]\varepsilon'$ is frequent in $\mathcal{S}$, then τ should be contained in $\mathcal{D}$, but constraints $\varepsilon[\tau^-, 1 + \tau^+]\varepsilon'$, $\varepsilon[\tau^-, 2 + \tau^+]\varepsilon' \ldots$, should be present in $\mathcal{D}$ too, and so should be $\varepsilon[\tau^- - 1, \tau^+]\varepsilon'$, $\varepsilon[\tau^- - 2, \tau^+]\varepsilon'$, $\varepsilon[\tau^- - 8, \tau^+ + 8]\varepsilon' \ldots$ In order not to store the infinite set of temporal constraints, the issue of building a complete constraint-database is to store only

those who *strictly occur* in $\mathcal{S}$. τ is said to *strictly occur* in $\mathcal{S}$ when its bounds cannot be tightened without changing its set of occurrences in $\mathcal{S}$. For instance, if we denote as $\mathcal{O}^{all}(\mathcal{C}, \mathcal{S})$ the set of all occurrences of $\mathcal{C}$ in $\mathcal{S}$, it appears that $A[-1,5]B$ *strictly occurs* in $\mathcal{S}_0$ while $A[-1,4]B$ and $A[0,5]B$ do not, since:

$$\begin{aligned}
\mathcal{O}^{all}(A[-1,5]B, \mathcal{S}_0) &= \{(A,1)(B,4), (A,1)(B,6), (A,5)(B,4), (A,5)(B,6)\} \\
\mathcal{O}^{all}(A[-1,4]B, \mathcal{S}_0) &= \{(A,1)(B,4), (A,5)(B,4), (A,5)(B,6)\} \\
&= \mathcal{O}^{all}(A[-1,3]B, \mathcal{S}_0) \\
\mathcal{O}^{all}(A[0,5]B, \mathcal{S}_0) &= \{(A,1)(B,4), (A,1)(B,6), (A,5)(B,6)\} \\
&= \mathcal{O}^{all}(A[1,5]B, \mathcal{S}_0)
\end{aligned}$$

$A[0,5]B$ and $A[1,5]B$ are said to be *equivalent* against $\mathcal{S}$, because they have the same occurrence-sets. We observe that $A[-1,4]B$ and $A[-1,3]B$ are equivalent too. Therefore the constraint $A[-1,5]B$ should be contained in $\mathcal{D}$ since we cannot tighten its bounds without changing the occurrence-set. However constraints $A[-1,4]B$ and $A[0,5]B$ should not be present in $\mathcal{D}$ since they both have a stricter constraint with the same occurrence-set. In other words, the problem of building the complete constraint-database is to keep in $\mathcal{D}$ only the strictest constraint of each class of equivalence.

To do that, we starts with the same strategy as in [3], except that we keep all frequent temporal constraints for each 2-episode $\varepsilon_1\varepsilon_2$, while [3] only keeps only one. Algorithm `CCDC` (alg. 3) details how the complete contraint-database is built.

Algorithm `CCDC` works as follows. For each pair of event types $(\varepsilon, \varepsilon')$ in $\mathbb{E}$, the set $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$ of all occurrences in $\mathcal{S}$ is processed (line 3). The notation $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$ stands for $\mathcal{O}^{all}(\varepsilon[-\infty,\infty]\varepsilon', \mathcal{S})$. For example in $\mathcal{S}_0$, the set of all event types is $\mathbb{E} = \{A, B, C\}$ and the occurrence-set for the pair (A, B) is $\mathcal{O}^{all}_{AB} = \{\langle(A,1)(B,4)\rangle, \langle(A,1)(B,6)\rangle, \langle(A,5)(B,4)\rangle, \langle(A,5)(B,6)\rangle\}$. For each occurrence in $\mathcal{O}^{all}_{AB}$, the gap between A and B is added to a set $\mathcal{A}_{AB}$ (line 4):

$$\mathcal{A}_{AB} = \texttt{sort}(\{3, 5, -1, 1\}) = \{-1, 1, 3, 5\}$$

Temporal constraints for the pair (A, B) are all obtained at lines 5-6 by iteratively sliding windows on $\mathcal{A}_{AB}$, whose widths are f_{th}, f_{th}+1, ..., $|\mathcal{A}_{AB}|$.

Algorithm 3 CCDC: Complete Constraint-Database Construction

Require: $\mathcal{S}$; $\mathbb{E}$; f_{th}
Ensure: $\mathcal{D}$
1: $\mathcal{D} \leftarrow \emptyset$
2: **for** each $(\varepsilon, \varepsilon') \in \mathbb{E} \times \mathbb{E}$ **do**
3: $\quad \mathcal{O}^{all}_{\varepsilon\varepsilon'} \leftarrow \{\langle(\varepsilon,t)(\varepsilon',t')\rangle | (\varepsilon,t) \in \mathcal{S}$ *and* $(\varepsilon',t') \in \mathcal{S}$ *and* $(\varepsilon,t) \neq (\varepsilon',t')\}$
4: $\quad \mathcal{A}_{\varepsilon\varepsilon'} \leftarrow$ **sort**$(\{(t'-t) | \langle(\varepsilon,t)(\varepsilon',t')\rangle \in \mathcal{O}^{all}_{\varepsilon\varepsilon'}\})$
5: $\quad$ **for** $k = f_{th}$ to $|\mathcal{A}_{\varepsilon\varepsilon'}|$ **do**
6: $\quad\quad \mathcal{K}^k_{\varepsilon\varepsilon'} \leftarrow \{\varepsilon[\mathcal{A}_{\varepsilon\varepsilon'}[i], \mathcal{A}_{\varepsilon\varepsilon'}[i+k-1]]\varepsilon' | 0 \leq i \leq |\mathcal{A}_{\varepsilon\varepsilon'}| - k + 1\}$ $\{\mathcal{A}_{\varepsilon\varepsilon'}[i]$ refers to the i^{th} element in $\mathcal{A}_{\varepsilon\varepsilon'}.\}$
7: $\quad$ **end for**
8: $\quad \mathcal{K}_{\varepsilon\varepsilon'} \leftarrow \mathcal{K}^k_{\varepsilon\varepsilon'} \cup \mathcal{K}^{k+1}_{\varepsilon\varepsilon'} \cup \ldots \cup \mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|}_{\varepsilon\varepsilon'}$
9: $\quad$ Transform $\mathcal{K}_{\varepsilon\varepsilon'}$ into the constraint-graph $\mathcal{G}_{\varepsilon\varepsilon'}$
10: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{G}_{\varepsilon\varepsilon'}$
11: **end for**
12: **return** $\mathcal{D}$

For instance with $f_{th} = 2$: ($\mathcal{K}^i_{AB}$ denotes the set of constraints that are obtained by sliding a window of width i on $\mathcal{A}_{AB}$)

$$\text{window's width} = 2\text{: } \mathcal{K}^2_{AB} = \{A[-1,1]B, A[1,3]B, A[3,5]B\}$$
$$\text{window's width} = 3\text{: } \mathcal{K}^3_{AB} = \{A[-1,3]B, A[1,5]B\}$$
$$\text{window's width} = 4\text{: } \mathcal{K}^4_{AB} = \{A[-1,5]B\}$$

By finally making the union of all $\mathcal{K}^k_{AB}$ (line 8), we obtain the set $\mathcal{K}_{AB}$ of all temporal constraints based on (A, B) that are frequent in $\mathcal{S}$. Line 9 adds arrows to $\mathcal{K}_{AB}$ to make the constraint-graph $\mathcal{G}_{AB}$. The algorithm that makes the constraint-graph $\mathcal{G}_{\varepsilon\varepsilon'}$ from $\mathcal{K}_{\varepsilon\varepsilon'}$ at line 9 is quite simple since we already know that $\mathcal{G}^{\top}_{\varepsilon\varepsilon'}$ is the unique element in $\mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|}_{\varepsilon\varepsilon'}$. Then it takes $\mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|-1}_{\varepsilon\varepsilon'}$ and adds an arrow from $\mathcal{G}^{\top}_{\varepsilon\varepsilon'}$ to each constraint in $\mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|-1}_{\varepsilon\varepsilon'}$. Then, the algorithm takes $\mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|-2}_{\varepsilon\varepsilon'}$ and searches for constraints τ' in $\mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|-2}_{\varepsilon\varepsilon'}$ that are stricter than any constraint τ in $\mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|-1}_{\varepsilon\varepsilon'}$ and adds an arrow from τ to τ' if so. Then, it takes $\mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|-3}_{\varepsilon\varepsilon'}$ and searches for constraints in $\mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|-3}_{\varepsilon\varepsilon'}$ that are stricter than any constraint in $\mathcal{K}^{|\mathcal{A}_{\varepsilon\varepsilon'}|-2}_{\varepsilon\varepsilon'}$, and so on until $\mathcal{K}^{f_{th}}_{\varepsilon\varepsilon'}$. Figure 2 shows the constraint-database we obtain when applying algorithm CCDC on $\mathcal{S}_0$.

The frequency we discussed in that section is not the same frequency as the one based on the recognition algorithm CRS. Indeed, given a temporal constraint τ, the set $\mathcal{O}^{all}(\tau, \mathcal{S})$ as it has been considered is this section contains all occurrences of τ in $\mathcal{S}$, not only occurrences that are recognized by CRS. For instance:

$$\begin{aligned}\mathcal{O}^{all}(A[-1,5]B,\mathcal{S}_0) &= \{(A,1)(B,4),(A,1)(B,6),(A,5)(B,4),(A,5)(B,6)\}\\ &\Rightarrow f_{all}(A[-1,5]B)=4\\ \mathcal{O}^{CRS}(A[-1,5]B,\mathcal{S}_0) &= \{(A,1)(B,4),(A,5)(B,6)\}\\ &\Rightarrow f_{CRS}(A[-1,5]B)=2\end{aligned}$$

This is not an issue. Indeed, since $f_{CRS}(\tau) \leq f_{all}(\tau)$ for all temporal constraints τ, it is possible to apply `CRS` counting on constraints in $\mathcal{D}$ in order to remove from $\mathcal{D}$ those that are not frequent in $\mathcal{S}$ before running algorithm `HDA` on $\mathcal{D}$. This is a bit annoying, but we have to build $\mathcal{D}$ that way with f_{all} and not with f_{CRS}, since the property "$\tau \preceq \tau' \Rightarrow \mathcal{O}^{CRS}(\tau,\mathcal{S}) \subseteq \mathcal{O}^{CRS}(\tau',\mathcal{S})$" is false, while this property is true with $\mathcal{O}^{all}(\tau,\mathcal{S})$. This property is implicitely used at lines 5-7 when `CCDC` iteratively computes stricter constraints from $\mathcal{O}^{all}_{\varepsilon\varepsilon'}$. `CCDC` would not be correct if it was executed with $\mathcal{O}^{CRS}$ instead of $\mathcal{O}^{all}$.

Consequently, if a constraint $\varepsilon[\tau^-,\tau^+]\varepsilon'$ is f_{CRS}-frequent, it will be contained in $\mathcal{D}$, but some constraints in $\mathcal{D}$ might not be f_{CRS}-frequent. A solution to cope with that problem is to purge $\mathcal{D}$ after it has been generated with `CCDC` by counting with `CRS` every constraint in $\mathcal{D}$. However it is not necessary to purge $\mathcal{D}$ since if there is in $\mathcal{D}$ any constraint τ that is not f_{CRS}-frequent, then τ will be counted as non-frequent once by `HDA` (line 9) and added to `non_Freq` (line 13), and as a result τ won't be propagated much in the mining process.

5.2 Building a constraint-database for hybrid episodes-discovery

The user might not be interested in the plain ability of expressiveness provided by chronicles. Indeed, in some cases there might be no worth precisely defining strong constraints with numerical bounds on event types of a chronicle. In those cases, the user can still use the framework presented in this paper by applying algorithm `HDA` to a simpler constraint-database, where only temporal orders with no numerical constraints between event types are stored. This refers to the problem of discovering hybrid episodes from a sequence of events [4]. Not much research work has focused on this hybrid episodes-discovery problem, contrary to serial or parallel episodes-

discovery. A solution to the "hybrid" problem is to apply algorithm HDA on the constraint-database $\mathcal{D}$, defined as follows :

1. $\forall(\varepsilon, \varepsilon') \in \mathbb{E} \times \mathbb{E}$ such that $\varepsilon <_{\mathbb{E}} \varepsilon'$, $\mathcal{G}_{\varepsilon\varepsilon'} \in \mathcal{D}$,
   where $\mathcal{G}_{\varepsilon\varepsilon'} = \{\varepsilon[-\infty, +\infty]\varepsilon', \varepsilon[-\infty, 0]\varepsilon', \varepsilon[0, +\infty]\varepsilon'\}$;
2. $\forall\varepsilon \in \mathbb{E}$, $\mathcal{G}_{\varepsilon\varepsilon} \in \mathcal{D}$, where $\mathcal{G}_{\varepsilon\varepsilon} = \{\varepsilon[0, +\infty]\varepsilon\}$.

On $\mathcal{S}_0$, the hybrid contraint-database is the one shown on figure 4. Of course, this constraint-database could be pre-counted too before applying algorithm HDA in order to remove non-frequent constraints.

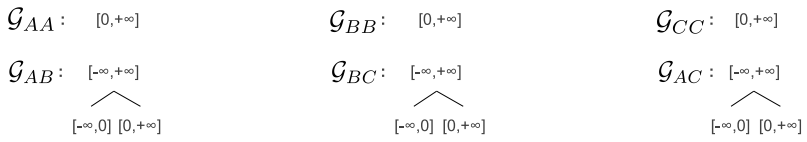


Figure 4: Constraint-database $\mathcal{D}_0$ to use when discovering hybrid episodes.

6 Discussion on perfomances and interactivity

6.1 Performances

We have previously estimated the time complexity of HDA as being lower than $O(p^{n_{max}^2} \times |\mathbb{E}|^{n_{max}})$, where n_{max} is the size of the biggest chronicle that is returned by HDA and p the size of each constraint-graph in $\mathcal{D}$. This complexity estimation is an upper bound of the number of chronicles that are processed

Biggest's size (n_{max})	Actual nb of candidates from	to	Total nb of candidates from	to	Ratio from	to
2	3	20	3	29	1	1
3	8	67	8	36696	1	632
4	54	1616	59	$4,23.10^7$	1	39133
5	579	21500	$1,16.10^8$	$3,40.10^{10}$	178050	$6,51.10^6$
6	6336	11831	$6,33.10^{11}$	$2,06.10^{14}$	$7,29.10^7$	$1,89.10^{10}$

Table 1: Actual number of chronicles processed compared to total candidate number of chronicles of size $n_{max} + 1$.

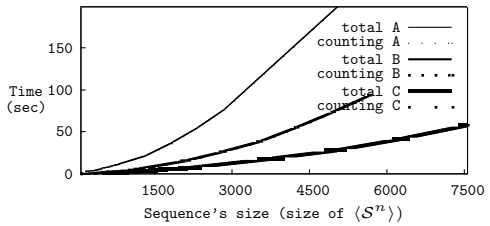
in iteration 3-23 and we argued that this estimation is in most cases far bigger than the actual number of chronicles that are processed, since whenever a chronicle is proved to be non-frequent none of its successors is added to `Open_chrs`. We tried to measure on real event sequences if this assumption makes sense.

All sequences we took are interaction traces of the smart kitchen [8]. Each interaction trace contains the sequence of all events that occurred when the user was preparing bruchettas. Events are ingredients and tools (spoon, knife, pan, bowl, etc.) that appear and leave the workbench. Changes to the oven's settings and to other appliance are also recorded. The chronicle discovery aims at learning task patterns in the user's activity that can be reused in order to improve task recognition in the smart kitchen and situation-aware assistance to the cook.

In total, there were 17 sequences, with $|\mathcal{S}|$ varying from 71 to 262, $|\mathbb{E}|$ varying from 36 to 42. We ran `HDA` on each of these sequences, with different values of f_{th} (f_{th} varying from 2 to 10). Therefore we performed 153 tries in total. Each time the mining time exceeded 30 seconds we stopped `HDA` and dropped the try, since we cannot predict it will end in a reasonnable time. Table 1 shows the ranges of the actual number of chronicles that have been processed by iteration 3-23 of algorithm `HDA`) compared to the ranges of the total number of chronicles of size $n_{max}+1$ for this try. For example, the line $n_{max}=4$ must be read as follows : tries where the biggest frequent minimal chronicle was of size 4 had an actual number of candidates ranging from 59 to $4,23.10^7$, while the total number of candidates of size 5 ($n_{max}+1$) ranged from $1,16.10^8$ to $3,40.10^{10}$. Depending on the try, there was from 1 to 39133 times less chronicles processed than the total number of candidates

Table 1 shows that the actual number of candidates processed is far lower than the total number of candidates. At first sight, the left column of table 1 suggests that when sequences are interaction traces of the smart kitchen the actual complexity is rather like $O(p^{n_{max}})$. Therefore, our assumption is true for the smart kitchen.

We could show many other measurements to investigate further the effects of `HDA`'s input parameters (f_{th}, $|\mathbb{E}|$, temporal dissemination of events in $\mathcal{S}\ldots$) on the execution time, but all measurements we made tended to prove that the only influential parameter was the resulting n_{max}, we cannot be known before mining the sequence. In order to address the scaling problem, we managed to increase the size of the sequence without impacting n_{max}. To do that, we applied `HDA` on $\mathcal{S}$ with f_{th}, then we concatenated $\mathcal{S}$ with $\mathcal{S}$ and



Seq $\mathcal{S}$	n_{max}	Size of $\langle \mathcal{S} \rangle$	Size of $\langle \mathcal{S}^n \rangle$
A	4	168	$168.n$
B	4	71	$71.n$
C	4	126	$126.n$

Figure 5: Time measures on three sequences when only the sequence's size varies. (processor intel Core 2 Duo 1,67Ghz - 2Gb RAM)

applied HDA on $\langle \mathcal{SS} \rangle$ with $f_{th} \times 2$ (we ensured that CRS couldn't recognize occurrences overlapping two consecutive sequences), then on $\langle \mathcal{SSS} \rangle$ with $f_{th} \times 3$, and so on. In that way, we make sure that the execution trace is exactly the same in each try, thus n_{max} too. We applied this strategies to three of our 17 sequences. Figure 5 shows the total time to mine the complete set of frequent minimal chronicles depending on the size of the concatenated sequence. We also measured the time spent by HDA on counting candidates chronicles. It's interesting to notice that both lines (total and counting time) are visually merged together, which means that most of the mining time is spent on counting tasks. As a result, the observed scaling behavior is the same as the one in the study of CRS [5, 6]. It also explains why we HDA makes use of Freq and non_Freq to limit the number of counts.

6.2 Coping with the complexity issue through interactivity

Previous section shows that the high complexity issue prevents from discovering big chronicles. This is an annoying issue since the user may expect that there may be big chronicles that are interesting to discover and that should come out of the discovery process. In the case of discovering task patterns in interaction traces, the user can cope with this issue by proceeding iteratively

as described in the interactive approach in [9]. HDA was actually designed to fit this approach. Indeed, HDA has two methods that can serve interactive purposes: is_acceptable and take_first_from. is_acceptable allows the user to push monotonic constraints into the mining process. For example, such constraints can be a maximum chronicle length, or an inclusion in a super chronicle. More complex and useful constraints can be found in existing works on interestingness measures [10, 11]. In the interactive approach, the user can first discover small chronicles by placing a *maximum length* constraint on HDA so that the mining does not take too much time. Then, for each "small" chronicle discovered a transformation replaces all occurrences of the chronicle in the interaction trace by a single event of a new event type, as described in [9]. At least, the user runs HDA again on the transformed sequence so as to discover new "small" chronicles that are composed of new event types, and so on iteratively. Each of these *compound chronicles* can be interpreted as a bigger chronicle. For instance, if $\mathcal{C}$ is composed of three event types, each of size 3, then $\mathcal{C}$ can be seen as a chronicle of size 9, while discovering chronicles of size 9 would have been probably much too big. The platform Trace Miner we developped (cf. figure 6) implements this interactive approach. On this example, Trace Miner is used to find relevant patterns in an interaction trace coming from the smart kitchen. The pattern "$chr_0 = \langle CKLN \rangle\ C[-8,5]K \ldots K[-5,4]N$" has been discovered by HDA and the user decided to open it in a chronicle editor (cf. part 3 of figure 6), probably to modify it and to store it to the file system or to add it as an inclusion constraint (see the contextual menu on part 3) for a later execution of HDA. The pattern add-board-knife (part 3') was discovered by HDA in an earlier execution of HDA. The pattern add-board-knife represents the situation of adding the cutting-board and the knife to the workbench. This pattern was actually found in almost all other cooking sequences and could be interpreted as the more abstracted user action "preparing the workbench to cut something".

The method take_first_from can have an important impact on the execution time. Indeed, first chronicles to be processed by HDA are the first chronicles to appear in Freq. We ran HDA with three different heuristics for take_first_from: LIFO, FIFO and *random choice*. We observed very different results depending on the heuristics. FIFO was by far the worst of them, since it performs a breadth-first search on the successor-graph (cf. figure 3). LIFO appeared to be more efficient than *random choice* in most cases, that is why results discussed in this paper were computed with LIFO. As Freq

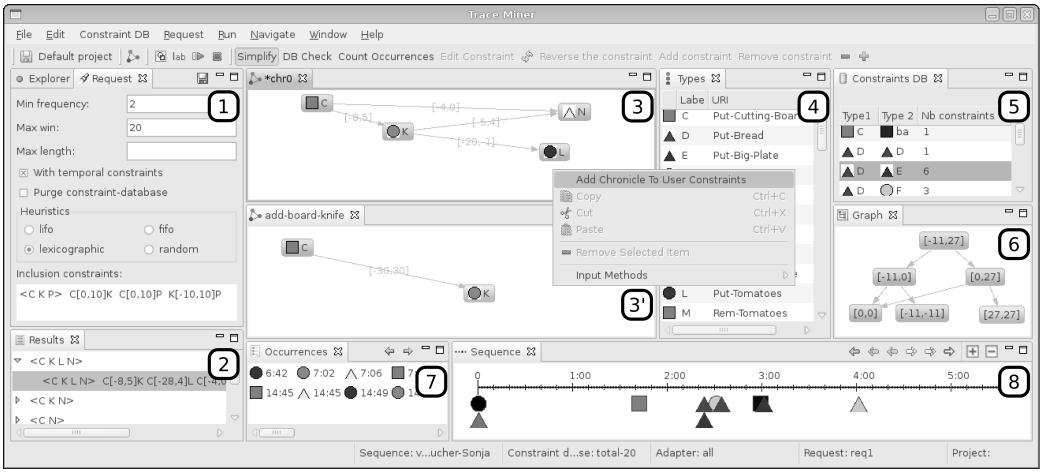


Figure 6: `Trace Miner`: `1` is the mining request, `2` the content of `Freq` in real time and `8` a navigation view of the mined sequence. `4` shows all event types and their associated shapes. `5` shows the list of all constraint-graphs in $\mathcal{D}$ and `6` shows the constraint-graph selected in `5`. 3 is a chronicle editor opened from the first result in `2`; $3'$ is a chronicle editor for the chronicle named `add-board-knife` that was stored to the file system. `7` shows the occurrences of 3 in the mined sequence showed in `8`. Inclusion constraints can be added by the user to the request view through the contextual menu of the chronicle editor or through drag and drop. `Trace Miner` also supports trace transformations based on chronicles. The view `Explorer` is unfortunately iconized on the figure and lists all resources (sequences, stored chronicles, transformations, etc.) that are used and created during the discovery process.

can make the user satisfy even though the mining process is not over, it is very important to put in `Freq` chronicles that are the most interesting for the user first. Interestingness measures can help with this challenging issue, but since they never reflect perfectly the user's real interest, we plan to drive `take_first_from` with user's knowledge. This user knowledge can be acquired by taking the opportunity of any user interaction with `HDA`, for instance when he adds a new constraint, to learn from the user's interests and expectations, as described in the interactive and opportunistic knowledge acquisition framework [12].

7 Conclusion

The framework presented in this paper contributes to the chronicle discovery problem by proposing a general approach that can be specialized in discovering the complete set of chronicles, or in discovering hybrid episodes, depending on the constraint-database that is given as input. In its complete form, this framework has a very high time complexity, but the actual execution time has been proved to be far shorter than the estimated one when applied on interaction traces. Furthermore, this framework allows the use of heuristics and user constraints so as to enable the discovery of big chronicles, despite its exponential complexity, by iteratively and interactively mining chronicles.

The next step of this work is to investigate further how user knowledge could be acquired and used to find chronicles that are of most interest for the user. Applied to interaction traces, the iterative and interactive discovery of interesting chronicles contributes to the trace abstraction issue [9]. The benefit of having abstracted interaction traces that describe the user's tasks with a high abstraction level is huge, especially when users need a meaningful support to share their experiences in a community like in the smart kitchen [8].

References

[1] Georgeon, O.: Analyzing traces of activity for modeling cognitive schemes of operators. In: AISB Quarterly. Number 127, 1-2 (2008)

[2] Faure, C., Delprat, S., Boulicaut, J.F., Mille, A.: Iterative bayesian network implementation by using annotated association rules. In: Proc. 15th Int. Conf. on Knowledge Engineering and Knowledge Management EKAW'06. LNAI, Springer (October 2006) 326–333

[3] Dousson, C., Duong, T.V.: Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In: IJCAI. (1999) 620–626

[4] Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery **1**(3) (1997) 259–289

[5] Dousson, C., Gaborit, P., Ghallab, M.: Situation recognition: Representation and algorithms. In: IJCAI. (1993) 166–174

[6] Dousson, C., Maigat, P.L., R&d, F.T.: Chronicle recognition improvement using temporal focusing and hierarchization. In: IJCAI. (2007) 324–329

[7] Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In Buneman, P., Jajodia, S., eds.: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. (1993) 207–216

[8] Schneider, M.: The semantic cookbook: Sharing cooking experiences in the smart kitchen. In: Proceedings of the 3rd International Conference on Intelligent Environments, Ulm , Germany., IET (2007) 416–423

[9] Cram, D., Fuchs, B., Pri, Y., Mille, A.: An approach to User-Centric Context-Aware Assistance based on Interaction Traces. In: MRC2008 : fifth International Workshop on Modeling and Reasoning in Context. (June 2008)

[10] Geng, L., Hamilton, H.J.: Interestingness measures for data mining: A survey. ACM Comput. Surv. **38**(3) (2006)

[11] Pei, J., Han, J., Wang, W.: Mining sequential patterns with constraints in large databases. (2002) 18–25

[12] Cordier, A., Fuchs, B., Lieber, J., Mille, A.: Failure Analysis for Domain Knowledge Acquisition in a Knowledge-Intensive CBR System. In: International Conference on Case-Based Reasoning, ICCBR'07. LNAI, Springer (2007) 463–477