# Efficient Mining of Temporally Annotated Sequences

Fosca Giannotti   Mirco Nanni
ISTI - CNR, Pisa, Italy
{f.giannotti, m.nanni}@isti.cnr.it

Dino Pedreschi
C.S. Dep., Univ. of Pisa, Italy
pedre@di.unipi.it

**Abstract**

Sequential patterns mining received much attention in recent years, thanks to its various potential application domains. A large part of them represent data as collections of time-stamped itemsets, e.g., customers' purchases, logged web accesses, etc. Most approaches to sequence mining focus on *sequentiality* of data, using time-stamps only to order items and, in some cases, to constrain the temporal gap between items. In this paper, we propose an efficient algorithm for computing *(temporally-)annotated sequential patterns*, i.e., sequential patterns where each transition is annotated with a *typical* transition time derived from the source data. The algorithm adopts a prefix-projection approach to mine candidate sequences, and it is tightly integrated with an annotation mining process that associates sequences with temporal annotations. The pruning capabilities of the two steps sum together, yielding significant improvements in performances, as demonstrated by a set of experiments performed on synthetic datasets.

## 1 Introduction

Frequent Sequential Pattern mining (FSP) deals with the extraction of frequent sequences of events from datasets of transactions; those, in turn, are time-stamped sequences of events (or sets of events) observed in some business contexts: customer transactions, patient medical observations, web sessions, trajectories of objects moving among locations.

As we observe in the related work section, time in FSP is used as a user-specified constraint to the purpose of either preprocessing the input data into ordered sequences of (sets of) events, or as a pruning mechanism to shrink the pattern search space and make computation more efficient. In either cases, time is forgotten in the output of FSP. For this reason, in our previous work [4] we introduced a form of sequential patterns annotated with temporal information representing typical transition times between the events in a frequent sequence. Such a pattern is called *Temporally-Annotated Sequence*, $\mathcal{TAS}$ in short.

In principle, this form of pattern is useful in several contexts: (i) in web log analysis, different categories of users (experienced vs. novice, interested vs. uninterested, robots vs. humans) might react in similar ways to some pages — i.e., they follow similar sequences of web access — but with different reaction times; (ii) in medicine, reaction times to patients' symptoms, drug assumptions and reactions to treatments are a key information.

In all these cases, enforcing fixed time constraints on the mined sequences is not a solution. It is desirable that typical transition times, when they exist, emerge from the input data.

The contributions of this paper are the following:

1. We provide a new algorithm for mining frequent $\mathcal{TAS}$, that is efficient and correct and complete w.r.t. the formal definition of $\mathcal{TAS}$– whereas the algorithm given in [4] provides approximate solutions.
2. We propose a new way for concisely representing sets of frequent $\mathcal{TAS}$'s, making them readable for the user.
3. We provide an empirical study of the performances of our algorithm, focusing on the overall computational cost and on some of the central and most interesting sub-tasks.

The paper is organized as follows: Section 2 provides an overview of related work and background information; Section 3 briefly summarizes the formal definition of the $\mathcal{TAS}$ mining problem; Section 4 describes in detail the proposed algorithm, and then Section 5 provides an empirical evaluation of the system. Finally, Section 6 closes the paper with some conclusive remarks.

## 2 Background and related work

In this section we summarize a few works related to the topic of this paper, and will introduce some relevant basic concepts and related works on sequential pattern mining, clustering and estimation of probability distributions.

**2.1 Sequence mining.** The *frequent sequential pattern* (FSP) problem is defined over a database of sequences $D$, where each element of each sequence is a

time-stamped set of items — i.e., an *itemset*. Timestamps determine the order of elements in the sequence. E.g., a database can contain the sequences of visits of customers to a supermarket, each visit being timestamped and represented as the set of items bought together. Then, the FSP problem consists in finding all the sequences that are *frequent* in $D$, i.e., appear as subsequence of a large percentage of sequences of $D$. A sequence $\alpha = \alpha_1 \rightarrow \cdots \rightarrow \alpha_k$ is a subsequence of $\beta = \beta_1 \rightarrow \cdots \rightarrow \beta_m$ ($\alpha \preceq \beta$) if there exist integers $1 \leq i_1 < \ldots < i_k \leq m$ such that $\forall_{1 \leq n \leq k}\ \alpha_n \subseteq \beta_{i_n}$. Then we can define the support $supp_D(S)$ of a sequence $S$ as the percentage of transactions $T \in D$ such that $S \preceq T$, and say that $S$ is frequent w.r.t. threshold $s_{min}$ if $supp_D(S) \geq s_{min}$.

Recently, several algorithms were proposed to efficiently mine sequential patterns, among which we mention PrefixSpan [8], that employs an internal representation of the data made of *database projections* over sequence prefixes, and SPADE [11], a method employing efficient lattice search techniques and simple joins that needs to perform only three passes over the database. Alternative methods have been proposed, which add constraints of different types, such as max-gap constraints and regular expressions describing a subset of allowed sequences. We refer to [13] for a wider review of the state-of-art on sequential pattern mining.

**2.2 Sequences with time.** In [10], Yoshida et al. propose a notion of temporal sequential pattern very similar to ours (see [4] or the summary provided in Section 3). It is called *delta pattern*, and integrates sequences with temporal constraints in the form of bounding intervals. An example of a delta pattern is the following: $A \xrightarrow{[0,3]} B \xrightarrow{[2,7]} C$, denoting a sequential pattern $A \rightarrow B \rightarrow C$ that frequently appears in the dataset with transition times from $A$ to $B$ that are contained in $[0,3]$, and transition times from $B$ to $C$ contained in $[2,7]$. While our work shares similar general ideas, the formulation of the problem is different, and this leads to different theoretical issues. However, Yoshida et al. simply provide an heuristics for finding *some* frequent delta patterns, do not investigate the problem of finding all of them, do not provide any notion of maximal pattern, and do not work out the theoretical consequences of their problem definition.

Another work along the same direction is [9], where an extension of delta patterns is proposed, with the name of *chronicles*. Essentially, a chronicle represents a general set of temporal constraints between events, whereas delta patterns were limited to sequential constraints. The former is represented as a graph, its vertices being events and its edges being temporal constraints between couples of events. As for delta patterns, constraints are represented as intervals.

Finally, several approaches can be found in literature for mining temporal patterns from different perspectives. Among the others, we mention the following: [6] defines temporal patterns as a set of states together with Allen's interval relationships, for instance "A before B, A overlaps C and C overlaps B"; [12] proposes methods for extracting *temporal region rules* of the form $E_C[a, b] \Rightarrow E_T$, meaning that any instance of condition $E_C$ is followed by at least one instance of $E_T$ between future $a$ and $b$ time scope. We refer also to [7] for a general overview of temporal phenomena in rule discovery.

**2.3 Probability distribution estimation and clustering.** Given a continuous distribution, estimating its probability density function (PDF) from a representative sample drawn from the underlying density is a task of fundamental importance to all aspects of machine learning and pattern recognition. As it will be clear from Section 3, also the mining of frequent *TAS*'s involves similar problems.

The most widely followed approaches to solve the density estimation problem can be divided into two categories: finite mixture models and kernel density estimators. In the former case we assume to be able to model the PDF as sum of a fixed number of simple components (usually normally distributed), thus reducing the PDF estimation problem to the parallel estimation of the statistics of each of the mixture components and their respective mixing weights. Kernel density estimators, on the contrary, estimate PDFs as a sum of contributions coming from all the available sample points — in some sense, it can be considered a mixture model with as many components as the number of sample points. Each sample point contributes with a different weight that is computed by applying a *kernel function* to the distance between the data point and the estimation point. Usually a distance threshold is introduced, called *bandwidth*, and data points beyond such distance give a null contribution. Each of these general estimation approaches gives rise to a family of clustering methods based on density estimation. Among the others, we mention the some of the most common ones: Expectation-Maximization (EM [2]) is a probabilistic model-based clustering method, i.e., a method which uses mixture models to estimate distributions; DENCLUE [5] and DBSCAN [3] are two clustering algorithms which estimate density by means of kernel functions (the former approach using any kernel function, the latter implicitly adopting a spherical, uniform one) and define clusters through *density-connectivity*: adjacent dense regions fall in the same cluster, thus yielding arbitrary-shaped clusters such that any two points of a

cluster are reachable traversing only dense regions.

## 3 Problem definition

In this section we briefly present the definition of TAS's and frequent TAS's, as described in [4]. As in the case of ordinary sequential patterns, frequency is based on a notion of sequence containment relationship which, in our case, takes into account also temporal similarity. Finally, we observe that frequent TAS's are in general too many (possibly infinite), and formalize our novel mining problem as the discovery of an adequate clustering of frequent TAS's.

DEFINITION 1. (TAS) *Given a set of items $\mathcal{I}$, a* temporally-annotated sequence of length $n > 0$, *called n–TAS or simply TAS, is a couple $T = (\bar{s}, \bar{\alpha})$, where $\bar{s} = \langle s_0, \ldots, s_n \rangle$, $\forall_{0 \leq i \leq n} s_i \in 2^{\mathcal{I}}$ is called the* sequence, *and $\bar{\alpha} = \langle \alpha_1, \ldots, \alpha_n \rangle \in \mathbf{R}_+^n$ is called the* (temporal) annotation. *TAS's will also be represented as follows:*

$$T = (\bar{s}, \bar{\alpha}) = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$$

EXAMPLE 1. *In a weblog context, web pages (or* pageviews*) represent items and the transition times from a web page to the following one in a user session represent annotations. E.g.:*

$$( \langle \{'/'\}, \{'/papers.html'\}, \{'/kdd.html'\} \rangle , \langle 2, 90 \rangle ) \\ \equiv \{'/'\} \xrightarrow{2} \{'/papers.html'\} \xrightarrow{90} \{'/kdd.html'\}$$

*represents a sequence of pages that starts from the root, then after 2 seconds continues with page 'papers.html' and finally, after 90 seconds ends with page 'kdd.html'. Notice that in this case all itemsets of the sequence are singletons.*

Similarly to traditional sequential pattern mining, we define a containment relation between annotated sequences:

DEFINITION 2. ($\tau$-CONTAINMENT ($\preceq_\tau$)) *Given a time threshold $\tau$, a n–TAS $T_1 = (\bar{s}_1, \bar{\alpha}_1)$ and a m–TAS $T_2 = (\bar{s}_2, \bar{\alpha}_2)$ with $n \leq m$, we say that $T_1$ is* $\tau$-contained *in $T_2$, denoted as $T_1 \preceq_\tau T_2$, if and only if there exists a sequence of integers $0 \leq i_0 < \cdots < i_n \leq m$ such that:*

1. $\forall_{0 \leq k \leq n}.\ s_{1,k} \subseteq s_{2,i_k}$
2. $\forall_{1 \leq k \leq n}.\ |\alpha_{1,k} - \alpha_{*,k}| \leq \tau$

*where $\forall_{1 \leq k \leq n}.\ \alpha_{*,k} = \sum_{i_{k-1} < j \leq i_k} \alpha_{2,j}$. As special cases, when condition 2 holds with the strict inequality we say that $T_1$* is strictly $\tau$-contained *in $T_2$, denoted with $T_1 \prec_\tau T_2$, and when $T_1 \preceq_\tau T_2$ with $\tau = 0$ we say that $T_1$* is exactly contained *in $T_2$. Finally, given a set of TAS's $D$, we say that $T_1$ is $\tau$-contained in $D$ ($T_1 \preceq_\tau D$) if $T_1 \preceq_\tau T_2$ for some $T_2 \in D$.*

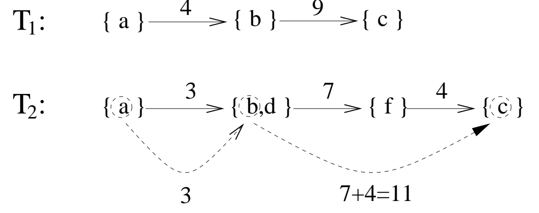



Figure 1: Example of $\tau$-containment computation

Essentially, a TAS $T_1$ is $\tau$-contained into another one, $T_2$, if the former is a subsequence of the latter and its transition times do not differ too much from those of its corresponding itemsets in $T_2$. In particular, each itemset in $T_1$ can be mapped to an itemset in $T_2$. When two itemsets are consecutive in $T_1$ but their mappings are not consecutive in $T_2$, the transition time for the latter couple of itemsets is computed summing up the times of all the transitions between them, which is exactly the definition of annotations $\alpha_*$. The following example describes a sample computation of $\tau$-containment between two TAS's:

EXAMPLE 2. *Consider two TAS's:*

$$T_1 = (\langle \{a\}, \{b\}, \{c\} \rangle, \langle 4, 9 \rangle) \quad and \\ T_2 = (\langle \{a\}, \{b, d\}, \{f\}, \{c\} \rangle, \langle 3, 7, 4 \rangle)$$

*also depicted in Figure 1, and let $\tau = 3$. Then, in order to check if $T_1 \preceq_\tau T_2$, we verify that:*

- *$\bar{s}_1 \subseteq \bar{s}_2$: in fact the first and the last itemsets of $T_1$ are equal, respectively, to the first and the last ones of $T_2$, while the second itemset of $T_1$ ($\{b\}$) is strictly contained in the second one of $T_2$ ($\{b, d\}$).*
- *The transition times between $T_1$ and its corresponding subsequence in $T_2$ are similar: the first two itemsets of $T_1$ are mapped to contiguous itemsets in $T_2$, so we can directly take their transition time in $T_2$, which is equal to $\alpha_{*,1} = 3$ (from $\{a\} \xrightarrow{3} \{b, d\}$ in $T_2$). The second and third itemsets in $T_1$, instead, are mapped to non-consecutive itemsets in $T_2$, and so the transition time for their mappings must be computed by summing up all the transition times between them, i.e.: $\alpha_{*,2} = 7 + 4 = 11$ (from $\{b, d\} \xrightarrow{7} \{f\}$ and $\{f\} \xrightarrow{4} \{c\}$ in $T_2$). Then, we see that $|\alpha_{1,1} - \alpha_{*,1}| = |4 - 3| < \tau$ and $|\alpha_{1,2} - \alpha_{*,2}| = |9 - 11| < \tau$.*

*Therefore, we have that $T_1 \preceq_\tau T_2$. Moreover, since all inequalities hold strictly, we also have $T_1 \prec_\tau T_2$.*

Now, frequent sequential patterns can be easily extended to the notion of *frequent TAS*:

DEFINITION 3. ($\tau$-SUPPORT, FREQUENT *TAS*) *Given a set $D$ of TAS's, a time threshold $\tau$ and a minimum support threshold $s_{min} \in [0,1]$, we define the $\tau$-support of a TAS $T$ as*

$$\tau\text{–}supp(T) = \frac{|\{T^* \in D \mid T \preceq_\tau T^*\}|}{|D|}$$

*and say that $T$ is* frequent *in $D$ if $\tau\text{–}supp(T) \geq s_{min}$.*

It should be noted that a frequent sequence $\bar{s}$ may not correspond to any frequent *TAS* $T = (\bar{s}, \bar{\alpha})$: indeed, its occurrences in the database could have highly dispersed annotations, thus not allowing any single annotation $\bar{\alpha} \in \mathbf{R}^n_+$ to be close (i.e., similar) enough to a sufficient number of them. That essentially means $\bar{s}$ has no *typical* transition times.

Now, introducing time in sequential patterns gives rise to a novel issue: intuitively, for any frequent *TAS* $T = (\bar{s}, \bar{\alpha})$, we can usually find a vector $\bar{\epsilon}$ of small, strictly positive values such that $T' = (\bar{s}, \bar{\alpha} + \bar{\epsilon})$ is frequent as well, since they are approximatively contained in the same *TAS*'s in the dataset, and then have very similar $\tau$-support. Since any vector with smaller values than $\bar{\epsilon}$ (e.g., a fraction $\bar{\epsilon}/n$ of it) would yield the same effect, we have that, in general, the raw set of all frequent *TAS* is highly redundant (and also not finite, mathematically speaking), due to the existence of several very similar — and then practically equivalent — frequent annotations for the same sequence.

EXAMPLE 3. *Given the following toy database of TAS's:*

$$a \xrightarrow{1} b \xrightarrow{2.1} c \quad a \xrightarrow{1.1} b \xrightarrow{1.9} c$$
$$a \xrightarrow{1.2} b \xrightarrow{2} c \quad a \xrightarrow{0.9} b \xrightarrow{1.9} c$$

*if $\tau = 0.2$ and $s_{min} = 0.8$ we see that $T = a \xrightarrow{1} b \xrightarrow{2} c$ is a frequent TAS, since $\tau$–supp(T)=1. However, we see that the same holds also for $a \xrightarrow{1.1} b \xrightarrow{2} c$ and $a \xrightarrow{1} b \xrightarrow{2.1} c$. In general, we can see that any $a \xrightarrow{\alpha_1} b \xrightarrow{\alpha_2} c$ is frequent whenever $\alpha_1 \in [1, 1.1]$ and $\alpha_2 \in [1.9, 2.1]$.*

A similar, more complex example is graphically depicted in Figure 2, where all frequent *TAS*'s for the sequence $\bar{s} = a \rightarrow b \rightarrow c$ over a toy dataset are plotted: the dataset is assumed to contain 10 transactions and each one contains exactly one occurrence of $\bar{s}$. The annotations of each occurrence are plotted as stars and called *dataset points*, adopting a terminology that will be introduced and better explained in later sections. Then, the darkest (blue) regions correspond to the infinitely many annotations $\bar{\alpha}$ that make $(\bar{s}, \bar{\alpha})$ a frequent *TAS* for $s_{min} = 0.3$ and $\tau = 0.1$; analogously, the lighter (green) shaded regions (plus the darkest/blue ones, implicitly) represent frequent *TAS*'s for $s_{min} = 0.2$,

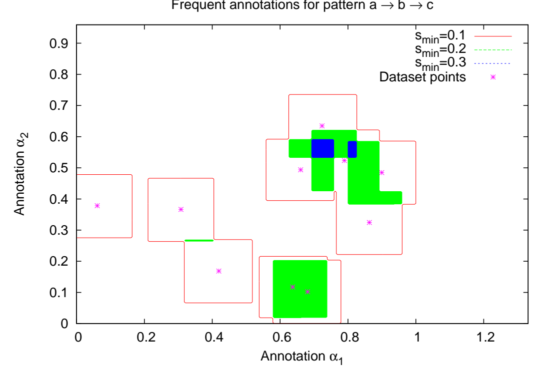



Figure 2: Sample dataset points and frequent annotations for $a \xrightarrow{\alpha_1} b \xrightarrow{\alpha_2} c$

and outlined regions correspond to frequent *TAS*'s for $s_{min} = 0.1$. Obviously enough, smaller $s_{min}$ values generate larger sets of frequent *TAS*'s and then correspond to larger regions in Figure 2.

A natural step towards a useful definition of frequent *TAS*'s, then, is the summarization of similar annotations (relative to the same sequence) through a single, concise representation.

The problem of discovering the frequent *TAS*'s for some fixed sequence can be formalized within a density estimation setting in the following way. Each sequence $\bar{s} = \langle s_0, \ldots, s_n \rangle$ can be associated with the space $\mathbf{R}^n_+$ of all its possible annotations, and so each *TAS* $T = (\bar{s}, \bar{\alpha}^*)$ $(\bar{\alpha}^* \in \mathbf{R}^n_+)$ *exactly contained* in some *TAS* of our database corresponds to a point in such space, that we can call a *dataset point*. Then, each annotation $\bar{\alpha} \in \mathbf{R}^n_+$ can be associated with a notion of frequency $freq(\bar{\alpha})$ that counts the dataset points close to $\bar{\alpha}$, more precisely defined as the number of dataset points that fall within a n-dimensional hyper-cube centered on $\bar{\alpha}$ and having edge $2\tau$. Figure 2 depicts a simple example with 10 dataset points (the stars) over $\mathbf{R}^2_+$ and $\tau = 0.1$ (notice the squares of side length $2\tau$ around each dataset point): dark regions represent annotations having frequency equal to 3; lighter regions correspond to frequency 2; finally, empty outlined regions contain annotations with frequency 1, while all the remaining points have null frequency.

We introduce a formal definition and notation for the two notions mentioned above:

DEFINITION 4. (DATASET POINTS, ANNOT. FREQ.) *Given a set $D$ of TAS's, an integer $n \geq 1$, a sequence $\bar{s}$ of $n+1$ elements and a threshold $\tau$, we define the set of* dataset points*, denoted as $A^n_{D,\bar{s}}$, as follows:*

$$A^n_{D,\bar{s}} = \{\bar{\alpha}^* \in \mathbf{R}^n_+ \mid (\bar{s}, \bar{\alpha}^*) \preceq_0 D\}$$

*and the* frequency *of any annotation $\bar{\alpha} \in \mathbf{R}^n_+$, denoted*

*as* $freq_{D,\bar{s},\tau}(\bar{\alpha})$, *as follows:*

$$freq_{D,\bar{s},\tau}(\bar{\alpha}) = \left|\{\bar{\alpha}^* \in A^n_{D,\bar{s}} \mid \|\bar{\alpha} - \bar{\alpha}^*\|_\infty \leq \tau\}\right|$$

*where* $\|\bar{\alpha} - \bar{\alpha}^*\|_\infty = \max_i |\bar{\alpha}_i - \bar{\alpha}^*_i|$.

We notice that such frequency notion considers all the possible (annotated) instances of a sequence in the database transactions and then, in general, differs from the $\tau$-support of $T = (\bar{s}, \bar{\alpha})$, since a $\mathcal{TAS}$ of the dataset can $\tau$-contain more than one instance of the same sequence $\bar{s}$ and thus can yield multiple annotations for $\bar{s}$. Viceversa, any number of instances of $\bar{s}$ appearing in different transactions and having exactly the same annotation would be mapped in $\mathbf{R}^n_+$ to the same point, thus contributing to $freq_{D,\bar{s},\tau}$ only as a single unit.

The notion of frequency described above, essentially corresponds to the estimated probability distribution that any kernel-based density estimation algorithm would compute on $\mathbf{R}^n_+$ from the set $\mathrm{A}^n_{D,\bar{s}}$ of all dataset points if it adopted a *uniform hypercube Parzen window* (sometimes simply called *Parzen window* or *naïve estimator*) as kernel function, with bandwidth $2\tau$, i.e., a kernel computed as product on $n$ independent uniform (i.e., constant-valued) univariate kernels having bandwidth $2\tau$ — which is equivalent to compute a normalized count of the elements contained in a hypercube with sides of length $2\tau$. Therefore, the problem of grouping frequent $\mathcal{TAS}$'s having similar annotations can be approximatively mapped to the problem of detecting dense regions on $\mathbf{R}^n_+$.

In the rest of the paper, we will present an algorithm for discovering and concisely represent frequent $\mathcal{TAS}$'s, that is based on the above described correspondence between frequency of $\mathcal{TAS}$'s in a dataset and density of annotations in an annotation space.

## 4 The algorithm

The pattern generation schema presented in this paper adopts and extends the PrefixSpan [8] projection-based method: for each frequent item $a$, a projection of the initial dataset $D$ is created, denoted as $D|_a$, i.e., a simplification which (i) contains only the sequences of $D$ where $a$ appears, (ii) contains only frequent items, and (iii) on each sequence the first occurrence of $a$ and all items that precede it are removed. In this case, the single-element sequence $a$ is called the *prefix* of $D|_a$. The fundamental idea is that any pattern starting with $a$ can be obtained by analyzing only $D|_a$, which in general is much smaller than $D$. Then, each item $b$ which is frequent in $D|_a$ will correspond to a frequent pattern $ab$ in $D$ (or $(ab)$, depending on how $b$ is added to the existing prefix), and a new, smaller projection $D|_{ab}$ (or $D|_{(ab)}$) can be recursively computed and used for finding longer patterns starting with $ab$ (or $(ab)$)[1].

In order to take full profit of the temporal constraints implicit in the definition of $\mathcal{TAS}$'s, prefix-projections are enriched with some information related to time and annotations. In particular, projected sequences are replaced by *T-sequences*:

Definition 5. (T-sequence) *Given a projected, time-stamped sequence* $S = \langle(s_1, t_1), \ldots, (s_n, t_n)\rangle$, *obtained as projection of sequence* $S_0$ *w.r.t. prefix* $s^*$ *(i.e.,* $S = S_0|_{s^*}$*), we define a T-sequence for* $S$ *as the couple* $(S, A)$, *where* $S$ *will be called the* temporal sequence, *and* $A = \langle(a_1, e_1), \ldots, (a_m, e_m)\rangle$ *is the* annotation sequence*: each couple* $(a_i, e_i)$ *represents an occurrence of the prefix* $s^*$ *in the original sequence* $S_0$, $a_i$ *being the sequence of time-stamps of such an occurrence*[2], *and* $e_i$ *being a pointer to the element of* $S$ *where the occurrence terminates, or the symbol* $\emptyset$ *if such element is not in* $S$. *Pointers* $e_i$ *will be called* entry-points.

As described in Section 3, given a sequence each transaction of the dataset is mapped into a set of annotations, corresponding to all the possible occurrences of the sequence in the transaction. *T-sequences* explicitly incorporate such information in the sequence, together with the exact point in the sequence where the occurrence ends.

Example 4. *Given the time-stamped sequence* $S = \langle(\{a\}, 1), (\{a, b\}, 2), (\{b, c\}, 3), (\{a\}, 4)\rangle$, *the T-sequence obtained for prefix* $a$ *will be the couple* $(S|_a, A)$, *where:*

$$S|_a = \langle(\{a, b\}, 2), (\{b, c\}, 3), (\{a\}, 4)\rangle$$
$$A = \langle(\langle 1\rangle, \emptyset), (\langle 2\rangle, \to 2), (\langle 4\rangle, \to 4)\rangle$$

*Here, for readability reasons, the notation* $\to 2$ *stands for "pointer to element having time = 2". The first occurrence of 'a' was* moved *into the prefix, so it does not appear in* $S|a$, *and therefore its corresponding pointer is set to* $\emptyset$. *In this case, only a single time-stamp appears in each element of the annotation sequence, since the prefix has length 1. Now, if we project* $S|a$ *w.r.t. b, we obtain prefix 'ab' and:*

$$S|_{ab} = \langle(\{b, c\}, 3), (\{a\}, 4)\rangle$$
$$A = \langle(\langle 1, 2\rangle, \emptyset), (\langle 1, 3\rangle, \to 3), (\langle 2, 3\rangle, \to 3)\rangle$$

*We notice that (i) now we have two time-stamps for each occurrence of the prefix; (ii) projecting w.r.t. '(ab)'*

[1] Hereafter, adopting quite standard conventions, we will call each itemset of a sequence an *element* of the sequence. Moreover, sequences are also represented as strings of items, with parenthesis around items in the same element, e.g.: $a(abc)(ac)$.

[2] Notice that annotation sequences contain time-stamps and not transition times (as opposed to $\mathcal{TAS}$'s): when needed, the latter are simply computed on-the-fly from the former.

*would yield a single occurrence having only one time-stamp, because the two items fall in the same element of the sequence; (iii) in the example the last two occurrences end at the same sequence location, but with different time-stamps, reflecting two different* paths *for reaching the same point.*

As it will be clearer at the end of this section, the advantage of using T-sequences is twofold:

- the annotation sequence corresponding to a prefix can be exploited to incrementally compute the annotation sequence of longer prefixes;
- the results of the frequent annotation search step can be exploited to eliminate some occurrences of the prefix (i.e., some elements of its annotation sequence). As an effect, it can (i) make faster the above mentioned computation of annotations, and (ii) allow, in the cases where all elements in the annotation sequence are deleted, to eliminate the whole T-sequence from the projection.

---

**Algorithm: MiSTA**
Input: A dataset $D_{in}$ of time-stamped sequences, a minimum support $s_{min}$, a temporal threshold $\tau$
Output: A set of couples $(S, \mathcal{D}^*)$ of sequences with annotations

1. $L = 0$, $\mathcal{P}_0 = \{D_{in} \times \{\langle\rangle\}\}$; *//Empty annotations*
2. **while** $\mathcal{P}_L \neq \emptyset$ **do**
3. $\mathcal{P}_{L+1} = \emptyset$;
4. **for each** $P \in \mathcal{P}_L$ **do**
5. **if** *P.length* $\geq 2$ **then**
6. $\mathcal{A}$= Extract_annotation_blocks($P$);
7. $\mathcal{D}$= Compute_density_blocks($\mathcal{A}$);
8. $\mathcal{D}^*$ = Coalesce_density_blocks($\mathcal{D}$);
9. $P^*$ = Annotation-based_prune($P$, $\mathcal{D}^*$);
10. Output $(P.prefix, \mathcal{D}^*)$;
11. **else** $P^* = P$; *//No annotations, yet*
12. **for each** item $i \in P^*$ **do**
13. **if** $P^*.enlarge_support(i) \geq s_{min}$ **then**
14. $\mathcal{P}_{L+1} = \mathcal{P}_{L+1} \cup \{$ enlarge_proj$(P^*, i)$ $\}$;
15. **if** $P^*.extend_support(i) \geq s_{min}$ **then**
16. $\mathcal{P}_{L+1} = \mathcal{P}_{L+1} \cup \{$ extend_proj$(P^*, i)$ $\}$;
17. $L$++;

---

Figure 3: Main algorithm for $\mathcal{TAS}$ mining

The overall algorithm is summarized in Figure 3. Steps 5–11 handle annotations, while all the others are essentially the same of PrefixSpan. In particular, steps 12–16 generate all sub-projections of the actual projection, separately performing *enlargement projections*, that add the new item to the last element of the prefix (therefore not changing the length of the sequence, but only making its last element one item larger), and *extension projections*, that add a new element to the prefix – a singleton containing only the new item. When a sub-projection $P$ is computed, some data structures used in the main program are updated:

- $P.prefix$: the prefix of projection $P$;
- *P.length*: the length of P's prefix, computed as number of elements;
- $P.enlarge_support(i)$: support of item $i$ within $P$, only counting occurrences of $i$ that can be used in a enlargement projection;
- $P.extend_support(i)$: support of item $i$ within $P$, only counting occurrences of $i$ that can be used in a extension projection.

Briefly, annotations are processed as follows: first (step 6), annotations are extracted from the projection, by scanning all annotation sequences, and their (hyper-cubical) areas of influence are computed; then (step 7), by combining them the space of annotations is partitioned into hyper-rectangles of homogeneous density, and such density is explicitly computed; therefore (step 8), such hyper-rectangles are merged together trying to maximize a quality criterium discussed in a later section; finally (steps 9-10), such condensed annotations are outputted and annotation sequences are filtered by eliminating all occurrences whose area of influence is not of any use in computing dense annotations. The steps listed above and the enlargement/extension projection procedure are discussed in detail in the following sections.

We remark that the objectives and results of density-based clustering differ from those of the density estimation task we are involved here. Indeed, the former focuses on partitioning the input set of points, and the density information is only a means for establishing a notion of *connectivity* between points. Therefore, existing density-based algorithms (and sub-space clustering algorithms in particular) cannot be applied for our purposes.

**4.1 T-sequences projection.** As already mentioned, our projections are composed of T-sequences, that differ from simple sequences in that they carry complete information on the location and annotation of each (useful) occurrence of the projection prefix in the sequence. That means, in general, that computing projections will require extra steps to keep annotation sequences up-to-date. That is especially true for extension projections, as summarized in Figure 4. In this case (steps 4–6), each annotation has to be extended

with each occurrence of the projecting item successive to the entry-point of the former – that becomes another *step* appended to the *path* described by the annotation element. That can be seen in Example 4 when projecting $S|_a$ w.r.t. $b$: the first annotation has a $\emptyset$ entry-point, and so it can be extended with both the occurrences of $b$ in $S|_a$, yielding two annotation elements with timestamps $\langle 1,2 \rangle$ and $\langle 1,3 \rangle$. The second annotation, instead, could be extended only with the second occurrence – the only one to be located after the entry-point (2). Finally, there is no occurrence after location 4, so the last annotation could not be extended at all.

This step has a $O(mn)$ complexity, $m$ being the number of occurrences of the item in the sequence, and $n$ being the length of the annotation sequence. In situations with an high repetition of the same item in a sequence, that can become a quite expensive task.

**Algorithm: extend_proj(P,i)**
Input: A projection $P$ and an item $i$
Output: A projection of $P$ w.r.t. $i$
1. $P' = \emptyset$;
2. **for each** T-sequence $T = (S, A) \in P : i \in T$ **do**
3. $S' = S|_i$ and $A' = \langle\rangle$;
4. **for each** annotation $(a, e) \in A$ **do**
5. **for each** $(s, t) \in S$ s.t. $i \in s \wedge t > e$ **do**
6. $A' = append(A', (append(a, t), \rightarrow t))$;
7. $P' = P' \cup \{(S', A')\}$;
8. **return** $P'$;

Figure 4: Extension projection procedure

The case of enlargement projections (Figure 5) is much simpler: for each annotation of the T-sequence to project, we just need to check if the sequence element $(s, t)$ pointed by the corresponding entry-point contains the projecting item $i$. Indeed, performing an enlargement projection w.r.t. $i$, essentially means to enlarge the last element of the projection prefix with $i$. Since $(s, t)$, for construction, already contains such last prefix element, we need to check only the presence of $i$. Therefore, the cost is simply linear in the length of the annotation sequence, reflected by the fact that step 5 here is a simple condition check, while in the extension projection a scan of (part of) the sequence was needed[3]. Notice that in case of positive result (step 6), the old annotation is simply kept unchanged.

[3]We remark that annotations make this kind of projection easier than what happens in the standard PrefixSpan algorithm: although the authors in [8] omit this detail, enlargement extensions in general would require a scan of the whole sequence, searching for an element that contains both the last element of the prefix and the projecting item.

**Algorithm: enlarge_proj(P,i)**
Input: A projection $P$ and an item $i$
Output: A projection of $P$ w.r.t. $i$
1. $P' = \emptyset$;
2. **for each** T-sequence $T = (S, A) \in P : i \in T$ **do**
3. $S' = S|_i$ and $A' = \langle\rangle$;
4. **for each** annotation $(a, e) \in A$ **do**
5. **if** $e$ points to element $(s, t) \in S$ and $i \in s$
6. **then** $A' = append(A', (a, e))$;
7. $P' = P' \cup \{(S', A')\}$;
8. **return** $P'$;

Figure 5: Enlargement projection procedure

**4.2 Extracting annotation blocks.** As discussed in Section 3, an annotation makes a sequential pattern frequent when it is similar to at least $s_{min}$ dataset points, i.e., annotations taken directly from the input data. Therefore, the general method adopted in this work for discovering frequent $\mathcal{TAS}$'s, given a sequence, is the following: (i) collect all dataset points and build their corresponding influence areas, i.e., the hyper-cubes centered in each dataset point and having edge $2\tau$; then, (ii) define the frequency (or support, or density) of an annotation as the number of such hyper-cubes it intersects; (iii) all areas (that, for construction, will have a hyper-rectangular shape) having frequency not smaller than $s_{min}$ are outputted as *frequent annotations*.

Now, as previously noticed, more than one dataset point can arise from a single input sequence while, on the other hand, the definition of $\tau$-support requires to count the number of matching *input sequences* – and not dataset points. In order to fix this mismatch, the algorithm builds the set of hyper-cubes as follows (see Figure 6): for each T-sequence in the projection, first (step 5) collect all its dataset points for the given sequence to annotate; then build the corresponding hyper-cubical influence areas and merge them (steps 6–7); finally, partition the resulting area into disjoint hyper-rectangles, and add them to the collection of influence areas (steps 8–9). This way, redundancy in the *area coverage* is eliminated, and each annotation that is *covered* by the processed T-sequence will intersect only one hyper-rectangle.

The "normalized" (disjoint) hyper-rectangles obtained in this step are called *annotation blocks*, and are the input for the successive steps of the algorithm, described in the next sections.

**4.3 Computing annotation densities.** In order to be able to discover and represent all the frequent annotations for a given sequence, we divide the annotation space in regions of homogeneous density, and select

**Algorithm: Extract_annotation_blocks(P)**
Input: A projection $P$
Output: A set of hyper-rectangles, representing the influence areas of each T-sequence in $P$.

1. $\mathcal{A} = \emptyset$;
2. **for each** T-sequence $T = (S, A) \in P$ **do**
3. $\quad \mathcal{A}_T = \emptyset$;
4. $\quad$ **for each** annotation $(a, e) \in A$ **do**
5. $\quad\quad$ Derive annotation $\bar{\alpha}$ from time-stamps $a$;
6. $\quad\quad$ $h$ = hyper-cube with center $\bar{\alpha}$ and edge $2\tau$;
7. $\quad\quad$ Merge $h$ with $\mathcal{A}_T$;
8. $\quad$ Partition $\mathcal{A}_T$ into a set of hyper-rectangles $\mathcal{A}'$;
9. $\quad \mathcal{A} = \mathcal{A} \cup \mathcal{A}'$;
10. **return** $\mathcal{A}$;

Figure 6: Extracting annotation blocks

**Algorithm: Compute_density_blocks($\mathcal{A}$)**
Input: A set of hyper-rectangles in $\mathbf{R}^d$
Output: A set of hyper-rectangles and their density.

1. $\mathcal{D} = \emptyset$;
2. Recursive_density($\mathcal{A}, d, \langle\rangle, \mathcal{D}$);
3. **return** $\mathcal{D}$;

**Algorithm: Recursive_density**($\mathcal{A}, d, \hat{H}, \mathcal{D}$)

1. $B = \{x | [l_1, h_1] \times \cdots \times [l_n, h_n] \in \mathcal{A}, x \in \{l_d, h_d\}\}$;
2. $\hat{B}$ =sorted_sequence($B$);
3. **for**$(i = 1; i < |\hat{B}|; i++)$ **do**
4. $\quad \mathcal{A}_i = \{[l_1, h_1] \times \cdots \times [l_n, h_n] \in \mathcal{A} \mid [l_d, h_d] \cap [\hat{B}_i, \hat{B}_{i+1}] \neq \emptyset\}$;
5. $\quad$ **if** $|\mathcal{A}_i| \geq s_{min}$ **then**
6. $\quad\quad \hat{H}' = append((l_d, h_d), \hat{H})$;
7. $\quad\quad$ **if** d=1 **then**
8. $\quad\quad\quad \hat{h} = [l_1, h_1] \times \cdots \times [l_n, h_n]$ given that $\hat{H}' = \langle (l_1, h_1), \ldots, (l_n, h_n) \rangle$;
9. $\quad\quad\quad \mathcal{D} = \mathcal{D} \cup \{\hat{h}\}$;
10. $\quad\quad\quad \hat{h}.density = |\mathcal{A}_i|$;
11. $\quad\quad$ **else**
12. $\quad\quad\quad$ Recursive_density($\mathcal{A}_i, d-1, \hat{H}', \mathcal{D}$);

Figure 7: Computing dense annotation blocks

those with a sufficiently high density. From an abstract viewpoint, that can be achieved by simply collecting the extreme coordinates of each annotation block along some dimension $d$, then split the space in correspondence of such values, and recursively re-apply the same process on the result for all the other dimensions. In Figure 7 we report an algorithm that performs such operation dimension by dimension, exploiting the known (decreasing) monotonicity property of density w.r.t. dimensionality of space. Thanks to this property, if a (segment of) space is not dense, all of its subspaces will be not dense: since we are looking for dense hyper-rectangles, in those cases we can safely stop the splitting process, avoiding to perform it along all the remaining dimensions.

More in detail, steps 1–2 set up the boundaries for the splits along the $d$-th dimension. Each boundary and the successive one locate an interval: step 4 performs a split for each interval, and step 5 checks the density threshold (equal to $s_{min}$). The splitting is recursively applied (step 12) to all intervals that pass the density check, until all dimensions have been split (step 7). In the latter case, all the intervals collected along the recursive calls of the procedure (see step 6) are combined to extract the hyper-rectangle they locate (step 8), which is added to the output ($\mathcal{D}$) and is associated with its density measure (steps 9–10). Figure 8(a) depicts a sample 2-dimensional result obtained by applying the algorithm to a synthetic dataset: darker regions represent higher densities, and the extracted *blocks* are artificially divided by white lines to better locate them.

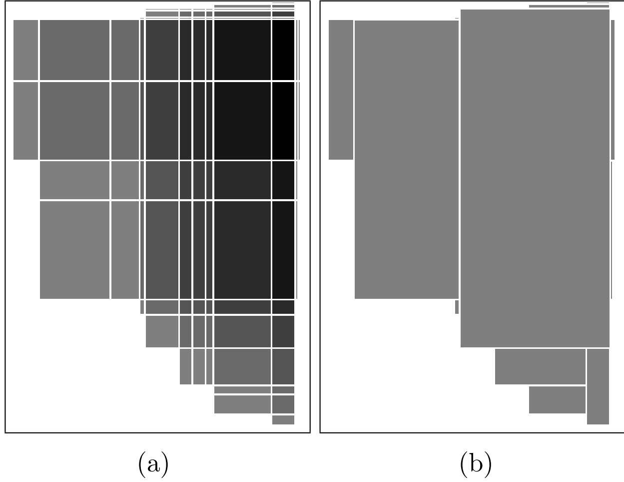
This algorithm has a high worst case complexity, essentially equal to $O(n^d)$, $n$ being the number of input annotation blocks and $d$ their dimensionality. However, the concrete cost of the method strongly depends on the overall density of the space searched. Empirical tests show that in several situations such cost remains under control, especially thanks to the fact that the maximal dimensionality of the space is typically limited, seldom above 10, and almost never above 20.

**4.4 Coalescing annotation densities.** The output of the task described in the previous section is a set of hyper-rectangles, each compactly representing a (infinite) set of frequent annotations (i.e., annotations that make the given candidate sequence a frequent $\mathcal{TAS}$). As implicitly shown in Figure 7 (steps 4 and 8 of the recursive procedure), the n-dimensional hyper-rectangles obtained by the algorithm can be nicely written as the cartesian product of $n$ 1-dimensional intervals: $h = [l_1, h_1] \times \cdots \times [l_n, h_n]$. Therefore, each output sequence $A_0 \to \cdots \to A_n$ with any of its dense hyper-rectangles $h = [l_1, h_1] \times \cdots \times [l_n, h_n]$ is essentially ready for presentation to the user in a compact, written form as follows:

$$A_0 \xrightarrow{[l_1,h_1]} A_1 \xrightarrow{[l_2,h_2]} \cdots \xrightarrow{[l_n,h_n]} A_n$$

meaning that *any* annotation having components within the specified intervals is frequent. In the rest of the paper we will call such writing a *elementary $\mathcal{TAS}$-set.*

However, the output of the Compute_density_blocks algorithm is typically composed of a large number of small hyper-rectangles, even in the cases in which they



(a) (b)

Figure 8: Sample result of Compute_density_blocks (a) and refinement obtained by Coalesce_density_blocks (b)

**Algorithm: Coalesce_density_blocks($\mathcal{D}$)**
Input: A set of dense hyper-rectangles
Output: A sequence of hyper-rectangles, covering the same volume as $\mathcal{D}$ but yielding a better approximation series.

1. $S = \emptyset$;
2. **while** $\mathcal{D} \neq \emptyset$ **do**
3. Select a random $h \in \mathcal{D}$ and let $\mathcal{D} = \mathcal{D} - \{h\}$;
4. **for each** extension direction $dir$ for $h$ **do**
5. $V_{dir}$ = volume of hyper-rectangle obtained by extending $h$ along direction $dir$;
6. $\mathcal{D}_{dir}$ = set of hyper-rectangles of $\mathcal{D}$ covered by the extension along $dir$;
7. **if** one extension was found **then**
8. $ext = \arg\max_{dir} V_{dir}$;
9. $h = h \cup \left(\bigcup_{h' \in \mathcal{D}_{ext}} h'\right)$;
10. $\mathcal{D} = \mathcal{D} - \mathcal{D}_{ext}$;
11. **goto** step 4.
12. $S = append(S, h)$;
13. **return** $\mathcal{A}$;

Figure 9: Coalescing dense annotation blocks

completely cover large regions of space. It is the case, for example, of the upper-right, darker colored section of Figure 8(a): there, a large rectangular region can be easily spotted, but it is divided into many sub-rectangles of slightly variable density. Outputting such result in a tabular format as shown above, would yield a long and quite unreadable list of patterns. The problem can be solved by performing some form of simplification of the structure of hyper-rectangles, trying to maximize some quality criterium.

In this paper we will follow a simple idea: the user should receive a sequence of elementary $\mathcal{TAS}$-sets to be interpreted as a series of successive approximations of the real set of dense annotations. Moreover, such approximation could disregard the exact density information, so that adjacent regions with different densities (yet always beyond the minimum support) could collapse if that yields a better result. Therefore, the dense hyper-rectangles obtained so far should be reorganized (merged and/or split) in order to build a sequence of elementary $\mathcal{TAS}$-sets such that the first outputted should provide the best possible approximation of the complete output, the second one should refine the approximation at the best, and so on. The key quality criterium we will adopt for evaluating an approximation is the coverage, i.e., the percentage of annotations (in a volumetrical sense) that is represented by the approximation.

Figure 9 shows a simple greedy algorithm that tries to solve the above mentioned problem. A initial dense hyper-rectangle is randomly chosen (step 3), and is repeatedly extended along the dimension and direction that yields the maximum increase in volume (steps 4–11). At each iteration, the hyper-rectangles covered this way are merged together and removed from the input rectangle set (steps 9–10). When no more extensions are possible, the obtained rectangle is added to the output (step 12). Then, the process is repeated on the remaining input rectangles, until all rectangles have been processed.

Figure 8(b) shows the result obtained by applying the method described above to the dense rectangle of Figure 8(a). As we can see, the dense region was highly simplified, reducing the number of rectangles from 89 to only 10. We notice, moreover, that there is no more the information on density, since heterogeneous rectangles were merged together. Alternative solutions may leave an approximate density information in the coalesced rectangles, such as the average, minimum or maximum density in the rectangle.

**4.5 Annotation-based projection pruning.** By knowing the dense annotations associated with a projection and its prefix, it is possible to divide the set of all the occurrences of the prefix in the projection into two categories: (i) the occurrences that contributed to form dense annotations, and (ii) those that did not. Each occurrence corresponds to a dataset point, that contributes to define the density in the annotation space within its hyper-cubical neighborhood. If no annotation within such neighborhood is dense, that means that such dataset point could have safely been disregarded in the density evaluation process, since no *interesting* region would have been affected, and no dense annotation would have been missed. Now, sub-projecting a pro-

jection means (also) to extend all its annotations by a temporal component – unless we are performing an enlargement projection, in which case annotations are left unchanged (see Section 4.1). Therefore, by means of projections all dataset points move from an annotation space to a higher-dimensional one: dense regions can become, in the new space, rarefied, while rarefied regions will always remain so. Therefore, dataset points that do not contribute to any dense region for a given projection/prefix, will always do the same for any extension of such prefix. That is to say, occurrences (i.e., their corresponding dataset points) that become *useless* at some stage of the computation, will remain useless till the end, and therefore can be safely eliminated before performing any new projection. Finally, we notice that a T-sequence that does not contain any *useful* occurrence of the prefix can only generate (larger) useless occurrences when projected. Therefore, when all dataset points of a T-sequence are eliminated, the T-sequence itself can be safely deleted.

The filtering process described above yields two main (positive) effects: (i) it reduces the number of dataset points to check in each T-sequence, making the successive annotation updates and density estimation processes faster; (ii) a number of T-sequences of the processed projection can be deleted, thus shortening the projection itself. In particular, if the projection remains with less than $s_{min}$ T-sequences, the projection process can be safely stopped, since no item can be frequent in such projection, and then no sub-projection could be performed later.

## 5 Experiments

In this section we show some experimental results obtained on synthetic datasets and aimed at assessing the performances of our algorithm. We will analyze the effects of input parameters on execution times, compare the performances with those of the PrefixSpan algorithm, and finally provide some details related to the extraction and treatment of dense annotations. Where not otherwise specified, experiments use datasets having around 100k sequences, $s_{min}$=0.5% and $\tau = 1$. The algorithm, implemented as a C++ command line application[4], was tested on a 2GHz Intel Xeon dual processor with 1GB RAM, running a RedHat 7.3 Linux operating system.

**5.1 Synthetic dataset generation.** In this work we extended the sequential pattern generator of [1] in order to enforce also typical transition times in the occurrences of the patterns. In particular, our generator associates each pattern with a random number of typical annotations, chosen between 1 and 4. Then, a time-stamp is assigned to each element of the generated dataset sequences. In particular, transition times between consecutive elements in each sequence vary from 0 to 1, assuming randomly distributed values. Then, each time the original data generator inserts a sequential pattern into a dataset sequence, the annotation associated with the pattern is perturbed by a normally distributed noise and the results are used as transition times of the actual instance of the pattern.

[4] Downloadable at `http://ercolino.isti.cnr.it/software`.

**5.2 Relative global performances.** As we can see in Figure 10, the algorithm scales almost linearly w.r.t. the dataset size and, as expected for any frequent pattern mining tool, performances quickly decrease when smaller minimum support thresholds are applied.

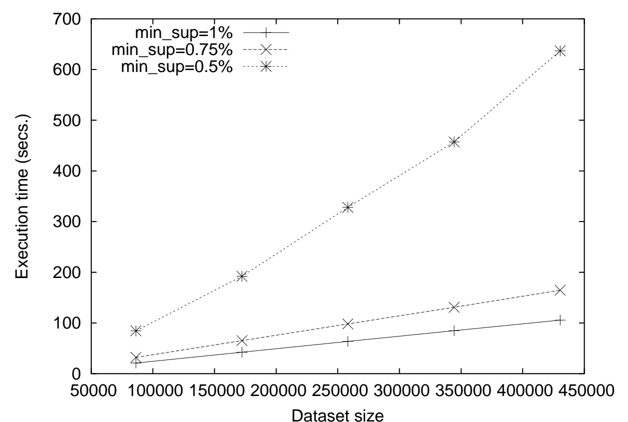



Figure 10: Exec. time vs. dataset size and $s_{min}$

Another key characteristic of data for sequential pattern algorithms is the average length of dataset sequences, which influences the number of frequent patterns present in the dataset, and, in our specific case, determines the quantity of annotations to be generated/updated at each projection. As Figure 11 shows, execution times quickly grow (apparently more than linearly) as the average sequence length grows, confirming our expectations.

**5.3 Comparison with PrefixSpan.** Being an extension of the PrefixSpan approach, it is natural to compare our algorithm with it. That is expected to clarify if and in which cases the overhead introduced by handling annotations is balanced by the additional pruning it allows. Experiments were performed by means of an implementation of PrefixSpan written by us, which later became the core prefix-projection engine at the base of our algorithm. Figure 12 summarizes the results of the comparison, where our algorithm was run with different values of the $\tau$ parameter.

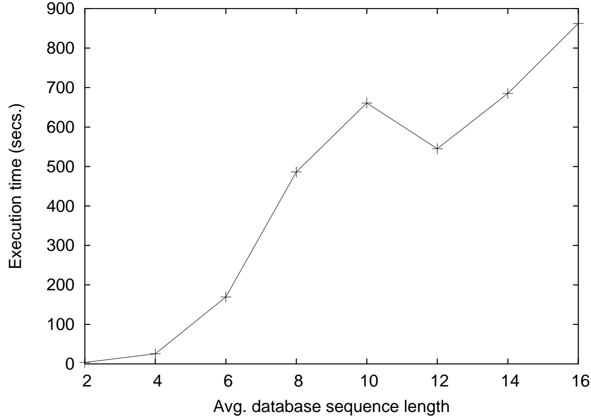



Figure 11: Exec. time vs. average sequence length

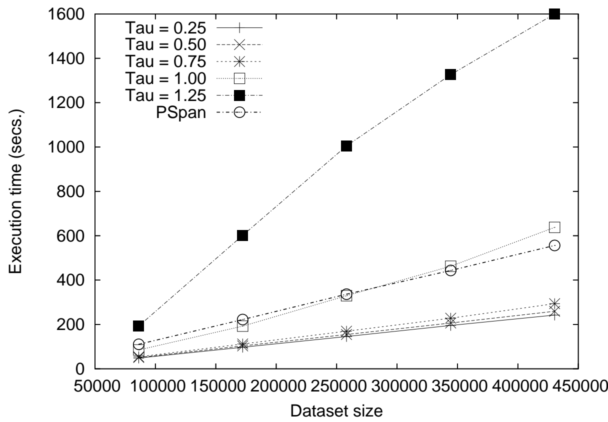



Figure 12: Comparison vs. PrefixSpan, with various $\tau$

The graph essentially says that for small values of $\tau$ the pruning power of annotations overcomes its overhead, while the opposite happens with larger values of $\tau$. That can be easily explained by observing that a large $\tau$ makes the area of influence of datapoints big, and therefore larger parts of the annotation space will be dense. As a consequence, a smaller number of dataset points will be eliminated, making the pruning strategy less effective. On the opposite, smaller values for $\tau$ produce a more rarefied annotation space, and therefore a larger quantity of dataset points become *useless*.

**5.4 Computing annotations.** Since the greatest novelty of our approach lies in the treatment of frequent annotations, special attention will be paid to the procedures that implement that aspect. In particular, we will analyze the cost of computing dense annotation blocks (Section 4.3) and the cost of simplifying them through coalescing (Section 4.4). Moreover, we will quantify both the fragmentation of the dense annotation blocks outputted by the first step, and the simplification power of the second one.

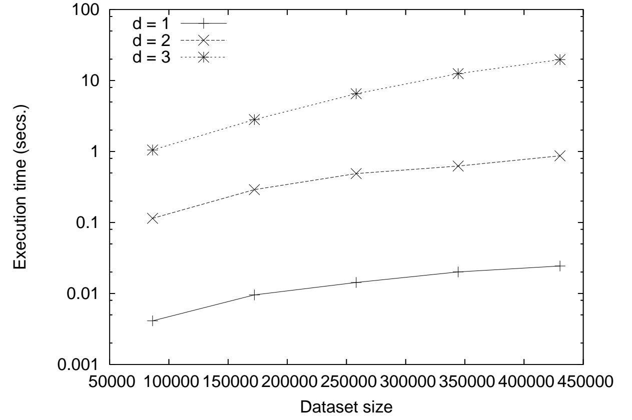



Figure 13: Computing density blocks

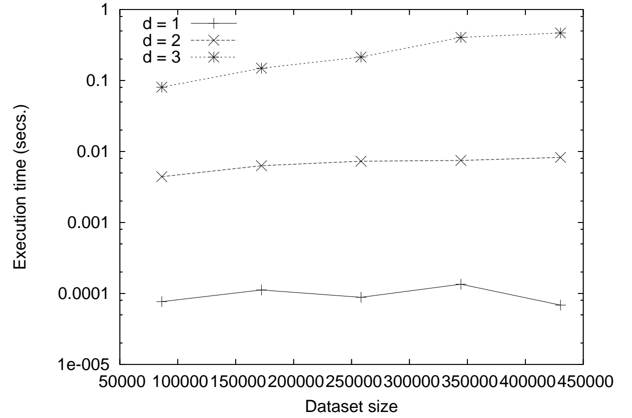



Figure 14: Coalescing density blocks

Figures 13 and 14 plot the average execution time of both tasks at three successive stages of the computation, i.e., the three curves in each plot represent the time required (on average) to discover dense annotations and to simplify them on a $d$-dimensional annotation space, with $d$ varying from 1 to 3. Comparing the two graphs (notice the log-scale on the vertical axis) we can derive that coalescing is a much cheaper operation than the discovery of dense annotation areas (around 1-2 orders of magnitude cheaper). However, the cost of both tasks grows quite smoothly with the size of the dataset, while they are heavily affected by the dimensionality of the annotation space – a unit increment in $d$ apparently causes a growth of an order of magnitute in execution times.

We end the experimental evaluation by measuring the average number of dense hyper-rectangles found for each size of the annotation space dimensionality.

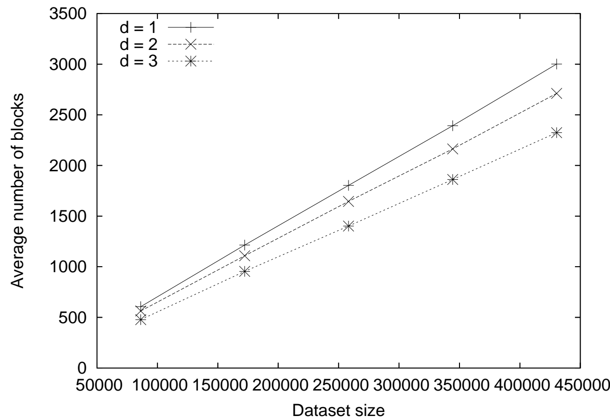



Figure 15: Raw density blocks generated per pattern

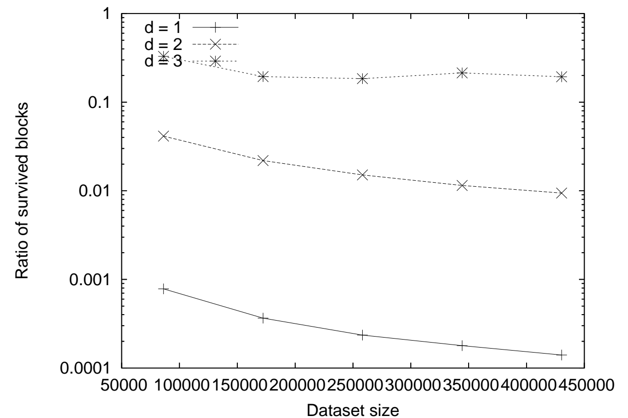



Figure 16: Survival ratio of blocks after coalescing

The objective is to give an idea of how fragmented the dense areas are before performing the coalescing step: the higher the dispersion, the more motivated is the coalescing step – although, in any case, it was shown to be a quite inexpensive task. As shown in Figure 15, the average number of (dense) hyper-rectangles generated for each pattern grows linearly with the dataset size, and increases faster on lower dimensions. In our experiments, such number of hyper-rectangles varies approximatively from 500 to 3000, and therefore is definitely too large for presenting the output to the user without any postprocessing.

Finally, we evaluate the impact of the coalescing procedure of the raw set of dense hyper-rectangles, by measuring the ratio of rectangles that survive the coalescing step. The smaller the ratio, the stronger the simplification power. Figure 16 clearly summarizes the results: the simplification power is very high on the 1-dimensional annotation spaces, but quickly shrinks at higher dimensionalities, yet always remaining at moderately good values – in our experiments, the survival ratio never exceeded 1/7. Moreover, the larger is the dataset, the stronger is the impact of coalescing.

## 6 Conclusions

In this paper we presented an efficient algorithm for computing frequent $\mathcal{TAS}$'s, based on the tight coupling of a prefix-projection strategy with a dense annotation discovery and pruning phase. Then, an experimental section was provided, that described in detail its behavior w.r.t. different parameters settings and different (synthetic) datasets.

The future work along this line of research includes several aspects, among which we mention the following: (i) validation on large, real datasets; (ii) low-level optimization of the algorithm, including support for parallel computation; and (iii) extension of the paradigm to other, non temporal-only, contexts, such as spatial and spatio-temporal ones.

## References


[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, 1995.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Stat. Soc.*, 39(1):1–38, 1977.

[3] M. Ester et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.

[4] F. Giannotti, M. Nanni, D. Pedreschi and F. Pinelli. Mining sequences with temporal annotations. To appear in ACM SAC 2006 – Data Mining track, 2006.

[5] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *KDD*, 1998.

[6] F. Höppner. Discovery of temporal patterns - Learning rules about the qualitative behaviour of Time Series. In *PKDD*, 2001.

[7] K. Morik. The Representation Race - Preprocessing for Handling Time Phenomena. In *ECML*, 2000.

[8] J. Pei et al. Prefixspan: Mining sequential patterns by prefix-projected growth. In *ICDE*, 2001.

[9] A. Vautier, M.-O. Cordier, and R. Quiniou. An inductive database for mining temporal patterns in event sequences. In *Processings of the workshop on Mining Spatial and Temporal Data*, 2005.

[10] M. Yoshida et al. Mining sequential patterns including time intervals. In *Procs. of SPIE Conf. - DMKD*, 2000.

[11] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[12] W. Zhang. Some Improvements on Event-Sequence Temporal Region Methods. In *ECML*, 2000.

[13] Q. Zhao and S. S. Bhowmick. Sequential pattern mining: A survey. Technical Report 2003118, Nanyang Technological University, Singapore, 2003.