Knowl Inf Syst (2015) 42:21–48
DOI 10.1007/s10115-013-0707-x

REGULAR PAPER

Fast time intervals mining using the transitivity of temporal relations

Robert Moskovitch · Yuval Shahar

Received: 27 February 2013 / Revised: 10 June 2013 / Accepted: 15 November 2013 /
Published online: 27 December 2013

Abstract We introduce an algorithm, called KarmaLego, for the discovery of frequent symbolic time interval-related patterns (TIRPs). The mined symbolic time intervals can be part of the input, or can be generated by a temporal-abstraction process from raw time-stamped data. The algorithm includes a data structure for TIRP-candidate generation and a novel method for efficient candidate-TIRP generation, by exploiting the transitivity property of Allen's temporal relations. Additionally, since the non-ambiguous definition of TIRPs does not specify the duration of the time intervals, we propose to pre-cluster the time intervals based on their duration to decrease the variance of the supporting instances. Our experimental comparison of the KarmaLego algorithm's runtime performance with several existing state of the art time intervals pattern mining methods demonstrated a significant speed-up, especially with large datasets and low levels of minimal vertical support. Furthermore, pre-clustering by time interval duration led to an increase in the homogeneity of the duration of the discovered TIRP's supporting instances' time intervals components, accompanied, however, by a corresponding decrease in the number of discovered TIRPs.

Keywords Temporal knowledge discovery · Temporal abstraction · Time intervals mining · Frequent pattern mining · Transitivity

R. Moskovitch (✉) · Y. Shahar
Department of Information Systems Engineering,
Ben Gurion University, Beersheba, Israel
e-mail: robertmo@bgu.ac.il

Y. Shahar
e-mail: yshahar@bgu.ac.il

R. Moskovitch
Telekom Innovation Laboratories, Ben Gurion University, Beersheba, Israel

R. Moskovitch
Department of Biomedical Informatics, Systems Biology and Medicine,
Columbia University, New York, NY, USA

1 Introduction

The increasing use and availability of longitudinal electronic data presents a significant opportunity to discover new knowledge from multivariate, time-oriented data, by using various data mining methods. Thus, multivariate temporal data mining is an important research topic within the data mining area, in which progress might have considerable beneficial implications. In particular, the effective discovery of frequent temporal patterns has significant potential benefits. Thus, in the medical domain, for example, it might lead to the *clustering* of patients who have a similar temporal pattern of interaction among the disease, a set of medications, and certain symptoms [5]. In other cases, it can lead to the *prediction* of certain outcomes, given, as features, certain temporal patterns in the patient's current longitudinal disease course. [4,19,24] In the information security domain, it might enable *classification* of hardware devices into infected and non-infected, by their temporal behavior [17].

However, temporal data include not only time-stamped raw data, or time *points* (e.g., hemoglobin value of 9.3 gr/100cc, at 9:05 am, on July 17th, 1998)), but also temporal *intervals*, possibly at a higher level of abstraction, which are either a part of the original raw input data (e.g., administration of a medication for 4 days), or are *abstractions*, or interpretations, derived from them as part of the computational process (e.g., two weeks of *moderate anemia*, or a month of *Grade II liver dysfunction*). We refer to such time intervals as *symbolic time intervals*. Several methods exist for abstraction of time-stamped data into symbolic time intervals; one option is to exploit domain knowledge [25], another is to use automated discretization methods [16–19].

However, regardless of the methodology used for the generation of the symbolic time intervals to support the multiple applications that use patterns composed of such time intervals (such as clustering, classification, and prediction), we need an effective method for the *discovery* of the frequent temporal patterns that are composed from the symbolic time intervals. Such an effective method is the main focus of the current paper. We refer to the resulting discovered knowledge regarding time-oriented concepts and their temporal relationships (patterns) as *temporal knowledge*.

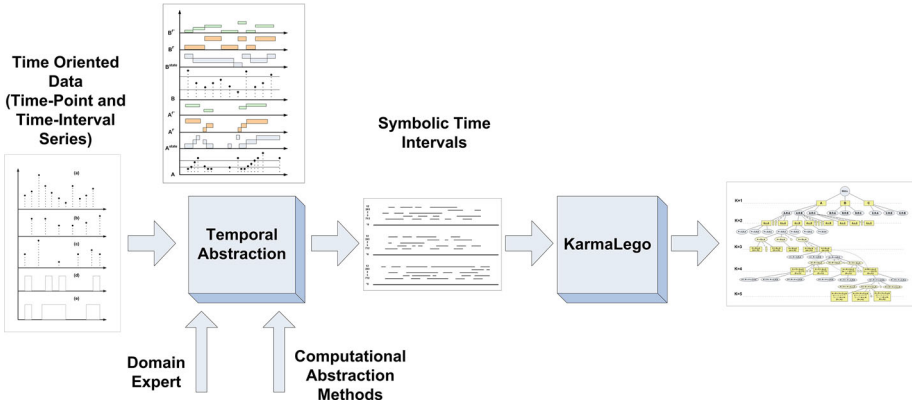

Fig. 1 The overall process of mining time interval- related patterns (TIRPs). The raw data time-point and time interval series are abstracted into a uniform format of [abstract] symbolic time intervals using domain knowledge or specialized computational means. The resulting symbolic time intervals are then mined and a tree of enumerated, sufficiently frequent temporal patterns is generated

The overall process that we envision is presented as a block diagram in Fig. 1. The input data include multiple instances of entities (e.g., patients or hardware devices) whose multiple variables (e.g., Hemoglobin value or Number of processes) are described by multivariate time-point series. The time-point series are abstracted, based on domain knowledge or on other computational means, and are transformed into symbolic time intervals (e.g., *Moderate Anemia* from t_1 to t_2). Then, the symbolic time intervals are mined using the KarmaLego algorithm, which is the focus of this paper, and which we introduce in Sect. 3 in detail, to discover frequently repeating temporal patterns.

As we explain in Sect. 3, the discovery process generates a tree of *Temporal Interval Relation Pattern*s (*TIRP*s); for each symbol, each branch represents a temporal pattern discovered in the data with a sufficient predefined frequency, which starts with the symbol represented at the root of the tree.

After a TIRP tree is discovered, several major application categories exist. These potential applications include:

- *Temporal knowledge discovery*, in which a domain expert manually and interactively reviews the automatically discovered temporal patterns for meaningful knowledge, using tools such as the KarmaLegoV visualization module [18].
- *Temporal clustering*, in which each temporal pattern is considered as a cluster of entities (e.g., patients, mobile devices) who have similar temporal behavior (e.g., a similar disease course or malfunction history). The use of the discovered TIRPs for the purpose of clustering is outside of the scope of the current paper. It is discussed in our study of clustering a population of diabetes patients by their disease course, or their temporal pathways [18], an example of which is shown in Sect. 4.2.
- *Prediction* rules, which are explicitly extracted based on the discovered temporal patterns and the transition probabilities between the components of the patterns.
- *Classification*, in which the discovered temporal patterns are used as features for a classification task, such as for implicit prediction of future outcomes using various data mining methods. The use of the discovered TIRPs for purpose of classification is outside of the scope of the current paper. It is discussed in our study of using TIRPs for the classification of the future outcomes of intensive-care unit patients [19].

The main contributions of the current paper can be summed up as follows:

1. Introduction in detail of the *KarmaLego algorithm for fast TIRPs mining,* which includes an efficient TIRPs data structure and a method for generating candidate TIRPs by direct extension of the evolving TIRP tree, which fully *exploits the transitivity of temporal relations*;
2. A rigorous *evaluation of the performance of the new framework* versus several existing algorithms, in several different time-oriented domains, demonstrating a significant speed-up compared with previous methods, especially with large datasets and low levels of minimal necessary TIRP support;
3. The use and evaluation of pre-clustering of the time intervals duration to reduce the variance in the supporting instances of the discovered TIRPs. This approach is proposed as complimentary for the non-ambiguous definition of TIRPs, which does not refer to the duration of the symbolic time intervals in a TIRP. Our experiments have demonstrated that pre-clustering by interval duration led to a decrease in the variance of the duration of the time intervals in the supporting instances, however, by a corresponding decrease in the number of discovered patterns.

In addition, the overall process we are using, shown in Fig. 1, built around the KarmaLego algorithm, includes several additional interesting features, which we explain in detail in the following sections:

a. The integration of *temporal abstraction* into the overall TIRP mining process;
b. The use of *an extended version of Allen's temporal relations* [1], in which flexible, more robust constraints are used to define the semantics of temporal relations;
c. The introduction of several fundamental concepts that define the output of a TIRP mining process, such as several measures of support across the overall subject population and within each subject, and a measure of variability of the discovered patterns.

2 Background

2.1 Temporal data mining

Temporal data mining [7,23] is a sub-field of data mining, in which various techniques are applied to time-oriented data to discover *temporal knowledge*, i.e. knowledge about relationships among different raw data and abstract concepts, in which the temporal dimension is treated explicitly. Unlike common data mining methods, which are static, often ignoring the temporal dimension, or using only concise statistical abstractions of it, temporal knowledge discovery presents significant computational and methodological challenges. However, temporal data mining offers considerable understanding of various scientific phenomena and the potential for creation of richer and accurate classification models, representing explicitly processes developing a long time.

2.2 Temporal abstraction

Temporal abstraction (*TA*) is the segmentation and/or aggregation of a series of *raw*, *time-stamped*, multivariate data into a *symbolic time interval* series representation, often at a higher level of *abstraction* (e.g., instead of a series of Hemoglobin or liver-function measurements, characterizations such as "3 weeks of moderate anemia," or "5 months of decreasing liver functions"), suitable for human inspection or for data mining.

Temporal abstraction, which typically includes also some form of interpolation [26], solves several common problems in mining raw time-series data, such as high variability in the sampling frequency and temporal granularity, minor measurement errors, and missing values, through the smoothing effect of the output abstractions. Thus, discovering frequent temporal patterns in multivariate temporal data can benefit from a preprocessing phase of converting the raw time-stamped data into a series of uniform, interval-based symbolic concepts. Figure 2 shows a TA process for one concept.

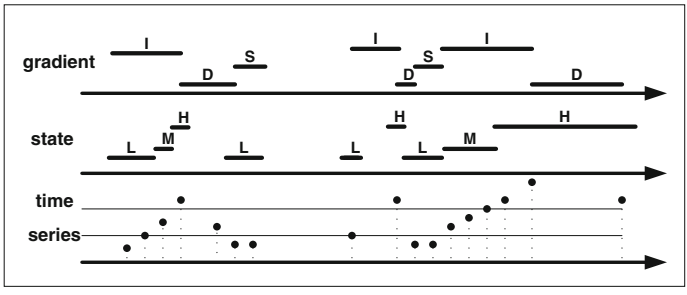

Fig. 2 A series of raw time-stamped data of one concept type (*at the bottom*) is abstracted into an interval-based *state* abstraction (a value classification) that has three discrete values: low (L), medium (M), and high (H) (*in the middle*); and into a *gradient* abstraction (the sign of the first derivative) that has the values increasing (I), decreasing (D), and stable (S) (*at the top*)

There are several approaches to the TA task; some exploit context-sensitive knowledge acquired from human experts, a method known as *knowledge-based temporal abstraction* (KBTA) [25]; others are purely automatic, and rely mostly on a discretization of the raw values and concatenation [3,9,12,14]. *Temporal discretization* refers to the process of discretization of a time-series values, usually performed through unsupervised means, as a preprocessing step in transforming the time-stamped, raw-concept series into a set of symbolic time intervals. [13]

2.3 Allen's temporal relations and transition tables

Allen formulated a finite set of 13 temporal relations between a pair of time intervals. The set includes: *before*, *meets*, *overlaps*, *starts*, *during*, *finishes*, and their corresponding inverse relations *after*, *met-by*, *overlapped-by*, *started-by*, *contains*, *finished-by*; and *equals* [1]. One might consider Allen's relations as being composed of seven basic relations, six of which have an inverse (*equals* is its own inverse), as shown in Fig. 5.

Allen's temporal relations were introduced for the purpose of temporal reasoning, such as for planning and narrative understanding, and since then were used for many purposes and were also further extended [6]. Allen also introduced the use of temporal relations *transition tables* for reasoning purposes, which was later generalized by Freksa by considering the option of having only semi-intervals [6].

Figure 3 shows four examples of reasoning about temporal relations based on their transitivity property. In all the cases there are three time intervals A, B and C. Given the temporal relations among A and $B - r^{A,B}$, and among B and $C - r^{B,C}$, we would like to reason about the possible relations among A and $C - r^{A,C}$.

In the first case, (1), $r^{A,B}$ is *before* ($<$) and the $r^{B,C}$ is *before* ($<$); thus, the only optional temporal relation for $r^{A,C}$ can be *before* ($<$). In the second case (2), $r^{A,B}$ is *overlap* (o) and $r^{B,C}$ is *before* ($<$); thus, $r^{A,C}$ is *before* ($<$) too. In the third case (3), $r^{A,B}$ is *meet* (m) and $r^{B,C}$ is *equal* (e), and thus, the only optional temporal relation which can be for $r^{A,C}$ is *meet* (m). Thus, in this scenario $r^{A,C}$ can be *before* ($<$), as it is actually in this illustration but it could be also *meet* (m) or *overlap* (o).

In Allen [1] and in [6] transition tables are presented in which, two temporal relations among three time intervals (among the first and second, and the second and third), the disjunction of the optional temporal relations among the first and third time intervals are

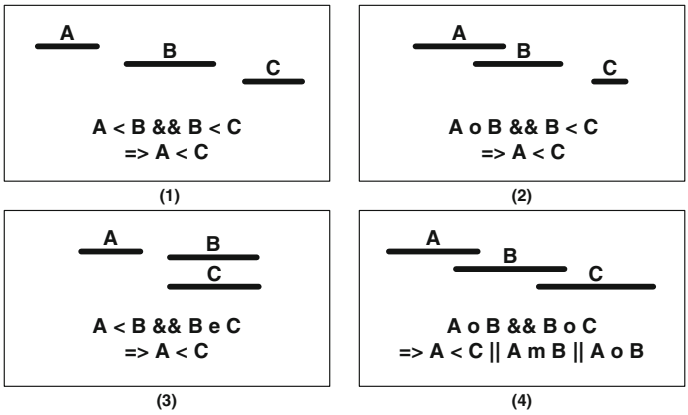

Fig. 3 Four cases of temporal transitivity, illustrating how the temporal relation $r^{A,C}$ can be derived, based on $r^{A,B}$ and $r^{B,C}$

stored. As we show in Sect. 3, one of KarmaLego's most novel contributions is in the use of transition tables for efficiently generating TIRP candidates.

2.4 Mining time intervals

Mining time intervals is a relatively young research field that has mostly sprung during the past decade. Most of the methods use some subset of Allen's temporal relations [1]. One of the earliest studies in the area is that of [27], which searches for containments of time intervals in a multivariate symbolic time interval series.

Kam and Fu [11] were the first to use all of Allen's temporal relations to compose time interval series. Their language covers the concepts of coincidence, synchronicity and order based on Allen's temporal relations. Their patterns are defined by the temporal relation of the extended frequent patterns with the new symbolic time interval, called $A1$ patterns. However, A1 patterns are ambiguous, since the temporal relations among the components of a pattern are undefined, except for the relations among all of the pairs of successive intervals.

Hoeppner [8] was the first to define a non-ambiguous representation of time interval patterns that are based on Allen's relations, by a k^2 matrix, to represent all of the pairwise relations within a k-intervals pattern. Using Allen's relations, [8,9] introduced a method inspired by association rules mining to mine rules in symbolic time interval sequences, restricted by using a sliding window.

In the rest of this paper, we shall refer to a conjunction of temporal relations between pairs of intervals as a time interval-related pattern (TIRP). The formal definition of a TIRP appears in Sect. 3 (Definition 5), as was defined and used already in [8,20]. Unlike Hoeppner's naïve mining method, [20] propose two approaches for generating the TIRP tree, using breath first search (BFS), in which the enumeration is made at each level before proceeding to the next level, depth first search (DFS), in which a greedy approach is used and each path is enumerated up to the leaves, and a hybrid approach, H-DFS, which combines the BFS and DFS methods, inspired by the Sequential Pattern Mining Algorithm (SPAM) [4] mining method. The BFS is initially used to discover the first two sized TIRPs in the mined database. Papapetrou et al. used only five temporal relations: meets, matches (equal, in terms of Allen's relations), overlaps, contains, and follows, similar to Allen's temporal relations. To make the temporal relations more flexible, Papapetrou et al. introduced an *epsilon threshold*, used only for a subset of the temporal relations, including meet, matches and follows relations.

ARMADA, by [28], is a relatively recent projection-based efficient time interval mining algorithm that uses a candidate generation and mining iterative approach. Wu et al. [29] proposed TPrefixSpan, which is a modification of the PrefixSpan sequential mining algorithm [22] for mining non-ambiguous temporal patterns from time interval events. TPrefixSpan represents the time intervals based on the start-time and end-time events and discovered frequent sequences of the start-time and end-time events from the projected database, although it does not prevent multiple candidates of the same patterns from being generated.

Patel et al. [21] introduced IEMiner, an algorithm that improves Papapetrou's method, by extending the patterns during the discovery process directly, while they are being considered. While several methods for non-ambiguous time intervals mining were proposed independently, not many runtime comparisons were made. Patel et al. [21] had compared their method runtime to TPrefixSpan [20,29] and found their method to be faster. In Sect. 4, we compare KarmaLego's runtime performance also to that of IEMiner [21], ARMADA [28], and H-DFS [20].

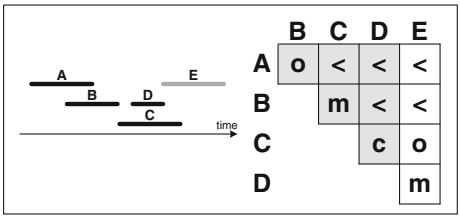

Fig. 4 An example of a time interval-related pattern (TIRP), represented by a sequence of five lexicographically ordered symbolic time intervals and all of their pairwise temporal relations. The indices inside the cells are part of the KarmaLego data structure (explained in Sect. 3.4). Note that symbolic interval E is being added to the four-symbol TIRP that includes A, B, C, D, thus extending it into a longer (candidate) TIRP; its temporal relations with the other four symbols appear at the right column of the temporal relations-half-matrix

Other options exist for enumeration of the temporal relations to be discovered. Mörchen [15] proposed an alternative to Allen's relations-based methods, in which time intervals are mined to discover partially ordered coinciding symbolic time intervals. Sacchi et al. [24] have used abstracted time series to find temporal association rules by generalizing several of Allen's relations into a relation called PRECEDES, which mainly included the temporal relation *before*.

The discovery of TIRPs is computationally highly demanding, since, in its most complete form, it requires generating all of Allen's seven basic temporal relations. For example, a naive generation of all TIRPs having 5 symbolic time intervals, such as in Fig. 4, with all possible temporal relations among them, requires in theory generating up to $7\hat{}((5^2-5)/2) = 7\hat{}10 = 282,475,249$ candidate TIRPs. In general, an extension of a k-sized TIRP, will result with $7\hat{}((k^2-k)/2)$ candidate TIRPs.

In the following section, we introduce the KarmaLego algorithm, designed to mine time intervals using all of Allen's temporal relations, all which are defined in a robust (flexible) fashion. Mining intervals using all Allen's temporal relations is very challenging from a computational perspective. Among other means, the KarmaLego mining method has an efficient data structure, and exploits the transitivity of the temporal relations for pruning, in order to efficiently generate all relevant candidate TIRPs.

3 Methods

3.1 KarmaLego—fast time intervals mining

To define formally the problem of mining time intervals, and to better understand the KarmaLego algorithm, we first present several basic definitions. These definitions will be used in the description of the methods.

3.2 Definitions

Definition 1 To define a flexible framework of Allen's temporal relations in KarmaLego, two relations are defined on time-stamped (point-based) data, given an epsilon value.

Given two time points t_1 and t_2:

$$t_1 = \varepsilon\, t_2 \; iff \; |t_2 - t_1| <= \varepsilon$$

and

$$t_1 < \varepsilon\, t_2 \; iff \; t_2 - t_1 > \varepsilon$$

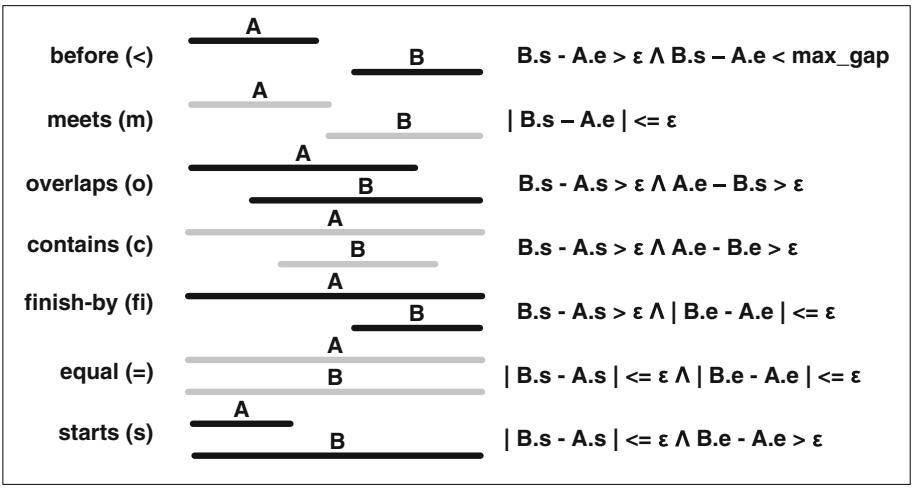

Fig. 5 A flexible extension of Allen's temporal relations, using the same epsilon for all of the seven basic relations

Based on the two relations $=\varepsilon$ and $<\varepsilon$ and the epsilon value, a flexible version of Allen's seven relations is defined, as shown in Fig. 5.

The introduction of the epsilon parameter to Allen's full set of temporal relations maintains the jointly exhaustive and pairwise disjoint (JEPD) conditions, as will be shown soon. The jointly exhaustive condition comes from probability theory and means that a set of events is jointly exhaustive if at least one of the events must occur. In the context of temporal relations, it means that the set of temporal relations, which are defined, must cover all of the optional relations among two time intervals.

The pairwise disjoint condition means that two sets A and B are disjoint if their intersection is the empty set. In the context of temporal relations, it means that the introduction of the epsilon value as defined in Definition 1 and Fig. 5 keeps the set of the temporal relations mutually exclusive. This is indeed true, since the epsilon-extended temporal-relation definitions appearing in Fig. 5 imply that for any two time intervals exactly one (epsilon-extended) temporal relation applies.

Definition 2 A *symbolic time interval*, $I = <s,\ e,\ sym>$, is an ordered pair of time points, start-time (s) and end-time (e), and a symbol (sym) that represents one of the domain's symbolic concepts. When referring to these properties we will use the following $I.s$ for its start-time, $I.e$ for its end-time, and $I.sym$ for its symbol.

Definition 3 A *symbolic time interval series*, $IS = \{I^1, I^2, \ldots, I^n\}$, where each I^i is a symbolic time interval, represents a series of symbolic time intervals, over each of which holds a start-time, end-time and a symbol.

Definition 4 A *lexicographical symbolic time interval series* is a symbolic time interval series, sorted in the lexicographical order of the start-time, end-time using the relations $< \varepsilon$, $= \varepsilon$ and the symbols, $IS = \{I^1, I^2, \ldots, I^n\}$, such that:

$$\forall I^i, I^j \in IS(i < j) \wedge \Big((I^i_{.s} <^{\varepsilon} I^j_{.s}) \vee (I^i_{.s} =^{\varepsilon} I^j_{.s} \wedge I^i_{.e} <^{\varepsilon} I^j_{.e}) \vee (I^i_{.s} =^{\varepsilon} I^j_{.s} \wedge I^i_{.e} =^{\varepsilon} I^j_{.e} \wedge I^i_{.sym} < I^j_{.sym}) \Big)$$

Since in our problem definition the time intervals are ordered lexicographically, we use only the seven temporal relations shown in Fig. 5.

Definition 5 A non-ambiguous lexicographical (TIRP) P is defined as $P = \{I, R\}$, where $I = \{I^1, I^2, \ldots, I^k\}$ is a set of k symbolic time intervals ordered lexicographically and

$$R = \bigcap_{i=1}^{k} \bigcap_{j=i+1}^{k} \Big\{ r(I^i, I^j) = \{r_{1,2}(I^1, I^2)\}, \ldots, r_{1,k}(I^1, I^k), \\ r_{2,3}(I^2, I^3), \ldots, r_{k-1,k}(I^{k-1}, I^k) \Big\}$$

defines all the temporal relations among each of the $(k^2 - k)/2$ pairs of symbolic time intervals in I.

Figure 4 presented a typical TIRP, represented as a half-matrix of temporal relations. We will usually assume such a representation through the description of the KarmaLego algorithm.

A problem with Definition 5 is that it ignores the precise duration of the time intervals that are the components of the TIRP. We focus on this problem, and on a possible solution of it, in Sect. 3.7, when we propose to pre-cluster the time intervals by duration.

Definition 6 Given a database of $|E|$ distinct entities (e.g., different patients), the *vertical support* of a TIRP P is denoted by the cardinality of the set E_P of distinct entities for which P holds, divided by the total number of entities (e.g., patients) $|E| : ver\_sup(P) = |E^P|/|E|$. The vertical support is actually what is commonly used as support in association rules, itemset and sequential mining.

When a TIRP has vertical support above a minimal predefined vertical support threshold, it is referred to as *frequent*.

We distinguish between two types of support: support across *all* entities that we call *vertical support* and within the longitudinal record of a *specific* entity, as will be introduced and defined in definition 7.

Definition 7 The *time intervals duration variance* (TIV) of a k-sized TIRP P, having n supporting instances, in which I^{Dur} is the duration of a time interval I ($I_{l,m}$ is the lth symbolic time interval (i.e., component) of the k time intervals of which instance m is composed) is:

$$\text{TIV}(P) = \frac{\sqrt{\sum_{l=1}^{k} \sum_{m=1}^{n} (I_{l,m}^{\text{Dur}} - \overline{I_l^{\text{Dur}}})^2}}{n \cdot k}$$

Where $\overline{I_l^{\text{Dur}}}$ is the mean of I_l^{dur}, that is, the mean duration of the lth component in all of the TIRP's n instances.

Definition 8 *The time intervals mining task*: Given a set of entities E, described by a symbolic time interval series IS, the goal of the time intervals mining task is to find all the TIRPs whose vertical support is above a predefined threshold.

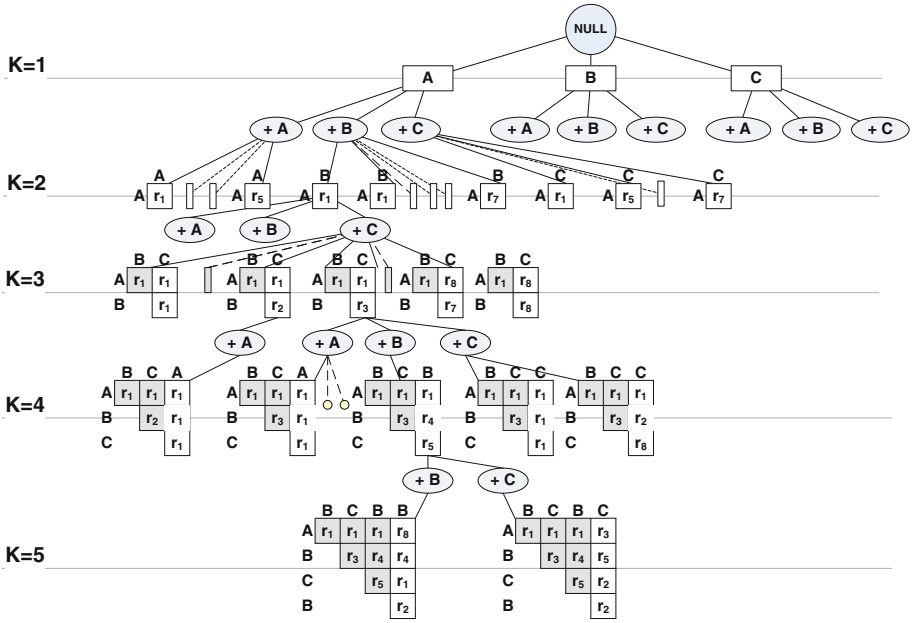

Fig. 6 A KarmaLego enumeration tree, in which the direct expansion of TIRPs is performed. Each *node* represents a TIRP as a half-matrix of the temporal relations. See text and detailed algorithms for an explanation

3.3 The KarmaLego algorithm

The KarmaLego (algorithm 1) consists of two main phases. The first phase is called Karma,[1] in which all of the frequent two-sized TIRPs, $r(I_{.1}, I_{.2})$ having two symbolic time intervals $I_{.1}$ and $I_{.2}$ that are ordered lexicographically and are related to r, a temporal relation, are discovered, and indexed. In the second phase, called Lego,[2] a recursive process extends the frequent 2-sized TIRPs, referred to as T^2, through efficient candidate generation, into a tree of longer frequent TIRPs consisting of conjunctions of the 2-sized TIRPs that were discovered in the Karma phase. Eventually a tree of all frequent TIRPs is discovered and returned (Fig. 6). Note that the KarmaLego algorithm is oblivious to the precise definition of temporal relations; we use the robust, flexible version introduced in Definition 1.

Algorithm 1 – KarmaLego

Input:
db – A database of |E| entities representing for each the symbolic time intervals of |S| symbols;
min_ver_sup – the minimal vertical support threshold;
Output: T – an enumerated tree of all frequent TIRPs

1. T ← Karma(db, min_ver_sup)
2. Foreach $t \in T^2$ // T^2 is T at the 2nd level
3. Lego(T, t, min_ver_sup)
4. End Foreach
5. return T
6. End

[1] Karma—The law of *cause* and *effect* originated in ancient India and is central to Hindu and Buddhist philosophies.

[2] Lego—A popular game, in which modular bricks are used to construct different objects.

3.4 Karma

In algorithm 2 (Karma), the first level of the enumeration tree is constructed and all the 2-sized TIRPs are discovered by indexing all the pairs of symbolic time intervals and their temporal relations, as described in lines 2–7. For each entity e (e.g., patient), the method incrementally indexes each pair of symbolic time intervals $I^i_{.sym}$, $I^j_{.sym}$; i and j represent their order in the symbolic time intervals series I ordered lexicographically in entity e; and sym is their symbol. After determining the temporal relation among the two symbolic time intervals r, based on their start-time and end-time, according to definition 1, the 2-sized TIRP instance is indexed in the proper node in the enumeration tree according to the symbols and the temporal relation.

Algorithm 2 – Karma

Input:
db – A database of |E| entities (the overall set of entities being referred to as E), representing for each entity e, the lexicographically sorted vector of its symbolic time intervals, e.I;
min_ver_sup – the minimal vertical support threshold;
Output: T – an enumerated tree of up to 2-sized frequent TIRPs

1. $T \leftarrow \emptyset$
2. Foreach $e \in E$
3. Foreach $I^i, I^j \in e.I \wedge i<j$
4. $r \leftarrow$ the temporal relation among I^i, I^j given *epsilon*
5. Index(T^2, < $e.I_{.sym}{}^i$, r, $e.I_{.sym}{}^j$>)
6. End Foreach
7. End Foreach
8. Foreach $t \in T^2$
9. if ver_sup(t) < min_ver_sup
10. Prune(t)
11. End Foreach
12. return T
13. End

After the indexing of all of the time interval pairs is completed, each t in T^2 that is not frequent is removed (pruned) from the tree. Eventually, an enumeration tree T is returned, at this point having only two levels. During the indexing process the first level of the enumeration tree is created as well, based on the frequency of the symbols.

In order to enable later a highly efficient retrieval of the 2-sized TIRPs for the recursive Lego step, T^2 is implemented by a two-dimensional square array of size $|T^1|^2$. $|T^1|$ is the number of the frequent symbols found in Karma. Each cell in the two-dimensional array contains a vector of size |R| (the number of temporal relations) that holds a HashMap, which contains all of instances of the indexed pairs. Thus, retrieval of the indexed pairs from T^2 in the Lego phase is performed in O(1), using as indices the two symbols and the temporal relation.

In the description of enumeration of the TIRPs using Lego we will use the following data structure, which implements a TIRP.

TIRP.sym—a vector of its ordered symbols;
TIRP.size—the size of the TIRP, defined as the number of the symbolic time intervals in the TIRP (|TIRP.sym|);
TIRP.tr—a vector that represents the half-matrix defining its conjunction of temporal relations; in Fig. 8, in each cell there is a number at the top-left corner, which is the index of the vector. Thus, the first index 0 represents the relation among the first and the second time intervals, the indices 1, 2 the relations among the first and second time intervals and

the third time interval, the indices 3–5 the relations among the first, second, and third time intervals, and the fourth time interval, and so on.
TIRP.tr_size— the size of the half-matrix defining the temporal relations in the TIRP, which is (TIRP.size^2 – TIRP.size)/2;
TIRP.insts—a list of the instances supporting the TIRP;
TIRP.inst.e_id—contains the entity id of the instance;
TIRP.inst.ti—a vector of pointers to the actual time intervals of the supporting instances. Note that the positions in all of the vectors of size k are indexed by the numbers $0 \cdots k-1$.

3.5 Candidate-TIRP generation: the supporting data structure

In KarmaLego, the candidate generation is made *directly*, i.e., by considering only specific frequent k-sized TIRPs and attempting to extend the ones being considered. Thus, extending a k-sized TIRP into a $(k+1)$ sized TIRP is made by adding a frequent symbolic time interval and adding all the possible temporal relations among the added symbolic time interval and the existing k time intervals, according to the lexicographical order in the TIRP.

More specifically, as shown in Fig. 7, the extension is done by generating for each frequent symbol a new symbolic time interval, and generating all of Allen's 7 temporal relations between the new symbolic time interval and the latest, in lexicographical order (see definition 5), symbolic time interval (i.e., the most recent addition) within the current extended TIRP. The last step is generating all of Allen's 7 temporal relations as the temporal relations among the new symbolic time interval and the existing TIRP's time intervals.

In Fig. 7, a 4-sized TIRP is extended by a symbolic time interval E that can be related to the TIRP's last time interval D (note that D is last according to the lexicographical order), having the latest start-time, by any of Allen's 7 temporal relations. The next step is generating, for each of these 7 possible temporal relations between E and D, all of the temporal relations among E and the previous time intervals: C, B and A. In this example having to set seven optional temporal relations in three temporal relations [r(A, E), r(B, E) and r(C, E)] would in theory generate up to $7^3 = 343$ candidates for each of Allen's 7 possible temporal relations, in addition to setting one of the seven temporal relations between E and D. Thus, overall, 7×343 candidates for the entire extension process. (later we will show how the 343 candidates can be decreased dramatically by exploiting the transitivity property). Finally, for each candidate TIRP, we must check whether all of its component relations are frequent and whether overall it is frequent. The extension process is implemented by the Lego algorithm.

Thus, Lego (Algorithm 3) receives a k-sized TIRP and extends it by all of the frequent symbols in T^1 and a set of predefined desired temporal relations R (Allen's seven relations, or other versions). First, all the frequent symbol candidates are generated and are added by relating them to the last time interval in t, using each of the temporal relations in R (lines 1

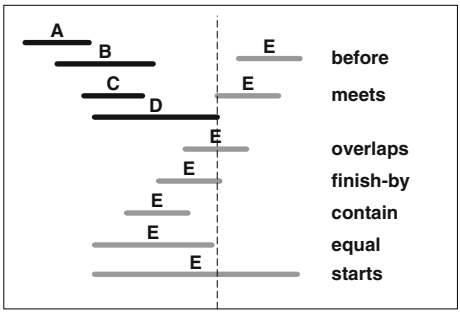

Fig. 7 A 4-sized TIRP (denoted by the *black intervals*) is directly extended by a symbolic time interval E, each time related to another temporal relation, among Allen's seven temporal relations, to the last symbolic time interval D

and 2). In line 3 a new TIRP is created of the extended TIRP, a symbol *sym* and a relation r are set to t^c in lines 4 and 5. C contains all of the candidates that were generated in line 7 by the method Generate Candidate TIRPs that will be described in the next section.

Then, for each candidate TIRP in C, a search is performed for its supporting instances (described in Algorithm 6), and if it was frequent, it is added to the current tree, and Lego extends the frequent candidate (recursively) in lines 8 to 13.

Algorithm 3 – Lego(T, t, min_ver_sup)
Input:
T – the enumeration tree created by Karma,
t – a TIRP that has to be extended,
min_ver_sup - the minimal vertical support threshold
Output: void

```
1. Foreach sym ∈ T^1
2.     Foreach r ∈ R
3.         Create new t^c of size (t.size + 1)
4.         t^c.s[t^c.size-1] ← sym
5.         t^c.tr[t^c.tr_size-1] ← r
6.         C ← 0
7.         C ← Generate_Candidate_TIRPs( t^c, 1)
8.         Foreach c ∈ C // candidates
9.             Search supporting instances(c, T^2)
10.            if( ver_sup(c) > min_ver_sup )
11.                T ← T ∪ c // c is frequent
12.                Lego(T, c, min_ver_sup)
13.        End Foreach
14.    End Foreach
15. End Foreach
16. End
```

The candidate generation in line 7 can be performed in two ways: a naïve candidate generation method, as will be presented in Algorithm 4, which is commonly used, or a faster method, exploiting a transition table, which is implemented in KarmaLego, as shown in Algorithm 5.

3.6 Efficient candidate-TIRP generation: exploiting transitivity in temporal relations

In line 3 of algorithm 2, an extended candidate TIRP t^c is created based on TIRP t, a symbolic time interval *sym* and a temporal relation r. Figure 8 illustrates an extension process of a 4-sized TIRP, including the symbolic time intervals: A, B, C, D, with a symbolic time interval E. The half matrix on the left presents the conjunction of the temporal relations defining the extended TIRP, in which each column contains the temporal relations among the symbolic time interval at the top and the earlier (lexicographically) time intervals in t on the left. The temporal relations, at the cells having the gray background, are the temporal relations defining t. For example, the temporal relation between B and D is $<$ (before). The rightest column contains the temporal relations among the new symbolic time interval (E) and the time intervals in t. The temporal relations that will be set in this column will generate the entire set of candidate TIRPs C.

Algorithm 4 generates (somewhat naively, as we shall see) the entire set of candidates C by first setting all the R temporal relations to the temporal relation between the new symbolic time interval and the latest time interval in the extended TIRP, and then generating all the combinations of the temporal relations in the rest of the temporal relations among the new symbolic time interval and the time intervals of TIRP t. For example, in Fig. 8, the temporal relation among the new symbolic time interval E and the latest D, shown in the cell having

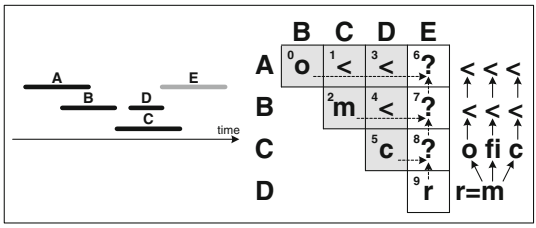

Fig. 8 The transitivity property is exploited in order to generate efficiently the possible temporal relations that are the candidate TIRPs. After setting the temporal relation between the new symbolic time interval E and the last (by lexicographical order) symbolic time interval D to *meets*, the possible temporal relations between E and the previous time intervals of the TIRP are inferred using a transition table. In this case, inferring by temporal transitivity decreased the number of potential candidates from 343 to only three. (The numbers in the left-top corner in the half-matrix cells refer to their index when represented by a vector in the algorithms)

the index 9 is set to the temporal relation *m* (meet). There are three temporal relations to generate (set), which total will be R^3; so for the case of using $R = 7$ temporal relations, there will be $7^3 = 343$ candidate TIRPs to be searched in T^2 (this is true with any relation among R set to index 9 and not specifically *meet*).

Algorithm 4 is recursive, and each time it creates for each temporal relation r in R a new candidate c^{new}, which is a copy of c (line 2), and r is set to the temporal relation between the new time interval and the *rIdx*+1 earlier time interval. When all the temporal relations were set, *rIdx* is smaller than the size of the TIRP and the generated TIRP is ready and added to C, otherwise the function is called again with c^{new}. Eventually the function returns the set of TIRP candidates C.

Algorithm 4 brute force candidate generation process results in a large number of candidates. The number of candidates grows exponentially with the number of time intervals in the TIRP. Thus, for example extending a 4-sized to a 5-sized TIRP (as the example in Fig. 8) generates $7^4 = 2401$ candidates, and extension from a 5-sized to a 6-sized TIRP will generate $7^5 = 16{,}807$ candidates for each added symbol.

However, note that such naïve generation results with many combinations of temporal relations that contradict each other and create an impossible TIRP. For example, consider a candidate based on Fig. 8, in which $\text{r}(D, E) = meet\text{s}$, and then the following [potential candidate] settings are made: $\text{r}(C, E) = meet\text{s}$, $\text{r}(B, E) = meet\text{s}$, $\text{r}(A, E) = meet\text{s}$. Obviously, setting $\text{r}(D, E)$ to *meet*s means that it is impossible to have $\text{r}(C, E) = meet$, since the temporal relation $\text{r}(C, D)$ is *contains*; similarly, the relations $\text{r}(B, E)$ and $\text{r}(A, E)$ contradict the existing relations.

Algorithm 4 - Gen_Candidate_TIRPs (Naively)

Input:

c – the base TIRP to generate

rIdx – the temporal relation index to generate

Output: C – set of the candidate TIRPs

1. Foreach r ∈ R
2. $c^{\text{new}} \leftarrow c$
3. c^{new}[c.tr_size-1-*rIdx*] ← r
4. if(c.size – *rIdx* > 2)
5. C← C ∪ Gen_Candidate_TIRPs (c^{new}, *rIdx*+1)
6. else
7. C← C ∪ c^{new}
8. EndForeach
9. return C
10. End

The ability to perform candidate generation without contradictions should decrease the number of candidates, and more important, should decrease the need to search for impossible candidates in T^2. To implement this capability, we use *transition tables* [1,6], which we introduced in Sect. 2.3. After setting the relation r(D, E) to *meets*, as shown in Fig. 8, the disjunction of the possible relations in r(C, E) can be reasoned based on r(C, D) and r(D, E) from a transition table, which results in this case with a disjunction of three possible temporal relations: *overlaps* (o), or *finished-by* (fi) or *contains* (c), as shown on the right side of Fig. 8. Similarly, the possible relations of r(B, E) can be computed based on r(B, C) and r(C, E). Since r(C, E) had a disjunction of three relations, the computation will be performed for each of the three temporal relations. Thus, when r(C, E) = o, using a transition table results in the conclusion that r(B, E) is *before* ($<$), and the same type of reasoning results in the cases, in which r(C, E) = fi, and r(C, E) = c. The same process is repeated to reason about r(A, E) for each of the temporal relations, which in this example were the same—*before* and result in the same relation, *before* as well. Thus, instead of $7^3 = 343$ candidates, there are actually only 3 candidates that do not contradict the TIRP's temporal relations. This process, which is implemented in Algorithm 5, obviously decreases dramatically the number of candidates that have to be searched in T^2 and the time that is required for that, as will be shown also empirically in the runtime evaluation results.

Algorithm 5 is a recursive process that exploits the transitivity property of temporal relations in order to generate candidates efficiently. The procedure gets the extended TIRP c and the temporal relation index *rIdx* to generate. The relation index (rIdx) starts as the temporal relation that was set in Lego (Algorithm 3) among the new, or next, symbolic time interval and the latest time interval, and ends with the last relation among the new symbolic time interval and the first time interval (for example, r(A, E) in Fig. 8).

Algorithm 5 - Gen_Candidate_TIRPs (with Transitivity)

Input:

c – the base TIRP to extend

rIdx – the temporal relation-index currently set

Output: c^{new} – the candidate TIRP

1. first_rel ←(((c.size-rIdx)^2- (c.size-rIdx))/2)-1
2. second_rel ← c.tr_size-rIdx
3. trans_rels ← trans_table[c.tr[first_rel], c.tr[second_rel]]
4. Foreach rel ∈ trans_rels
5. c^{new} ← c
6. c^{new}[scnd_rel-1] ← rel
7. if(first_rel > 0)
8. $C \leftarrow C \cup$ Gen_Candidate_TIRPs(c^{new}, rIdx+1)
9. else
10. $C \leftarrow C \cup c^{new}$
11. EndForeach
12. return C
13. End

Given the size of the TIRP *c.size* and *rIdx*, the indices of the first and second relations to be looked up in a transition table are calculated (lines 1,2) and the actual disjunction of temporal relations—*trans_rels*—is computed and returned by the transitive table at line 3. Then, for each of the relations, a new TIRP c^{new} is generated in line 5 and the relation is set in the appropriate location in the relations-half-matrix vector. If the index of the first relation, *frst_rel* is zero, then the generated set of candidate TIRPs c^{new} is added to C, otherwise *Get_Candidate_TIRPs* is called again with c^{new}. Eventually the function returns the set of

candidate TIRPs C. After the candidates C were generated, the supporting instances are searched in line 9 in *Lego* (Algorithm 2).

Algorithm 6 describes the method of searching for supporting instances of a potentially extended TIRP and adding them to the supporting instances list of the extended TIRP. The method searches for the next appropriate symbolic time interval(s) for whom the defined temporal relations with the previous symbolic time intervals in the extended TIRP hold. After executing this method, the vertical support of the new TIRP, which is a longer pattern than the one to be extended, will be equal to or (usually) lower than the extended TIRP's vertical support. (Note that it is possible, in the case of certain specific entities, that the number of occurrences of the extended TIRP within a particular entity's set of symbolic time intervals might actually grow, when compared with the original TIRP. This could happen when there are several suitable instances of symbolic time intervals within the same entity that can extend the same original TIRP instances.)

The method receives as input the candidate TIRP c and the set of the two sized TIRPs in T^2. In line 1, the next (new) symbol that was added (in Algorithm 3—Lego) is set to *next_sym*; then, for each instance in the extended TIRP's supporting instances, the search is made. First the temporal relation *rel* between the next time interval and the latest (in the extended TIRP) is set (line 3), then the latest symbol of the extended TIRP *sym* is set (line 4).

GetNextSTIs searches the symbolic time interval (STI) two- dimensional square array T^2, as explained in Sect. 3.4, using indices defined by the symbols *sym* and *last_sym* and the temporal relation *rel*, for the instances starting with the latest time interval in the instance *ins.sti*. *GetNextSTIs* might return several new symbolic time intervals, *next_stis*.

Algorithm 6– Search_Supporting_Instances

Input:
c – the TIRP to extend by searching supporting symbolic interval instances
T^2 – the 2-dimensional array of 2-sized TIRPs instances
Output: c – extended by the supporting instances

```
1. next_sym ← c.sym[c.size-1]
2. Foreach inst ∈ c.insts
3.   rel ← c.tr[c.tr_size-1]
4.   sym ← c.sym[c.size-2]
5.   next_stis←GetNextSTIs(inst.sti[c.size-2],T2[sym, rel, next_sym])
6.   Foreach next_sti ∈ next_stis
7.       inst^new ← inst
8.       inst^new.sti[c.size-1] ← next_sti
9.       For(i=1; i<c.size-1; i++)
10.          rel ← c.tr[c.tr_size-1-i]
11.          sym ← c.sym[c.size-2-i]
12.          if(NoInst?( inst^new.e_id, inst^new.sti[c.size-2-i], next_sti, T2[sym,rel,next_sym])
13.                break
14.      EndFor
15.      c.insts ← c.insts ∪ inst^new
16.  End Foreach
17.  remove inst from c.insts
18. End Foreach
19. return c
20. End
```

For each next symbolic time interval *next_sti*, the existence of all the temporal relations among *next_sti* and the earlier time intervals of the extended TIRP have to be checked in T^2 (lines 9–15). First, *inst* is copied into a new instance $inst^{new}$ (line 7), the next time interval

next_sti is set to its proper location (line 8), and then, using a loop, the pairs of the time intervals and the temporal relations are searched at the relevant T^2 (using as indices the current-TIRP's symbol *sym*, the next symbol *next_sym*, and the temporal relation *rel*) in lines 9–16. Thus, in line 9, the existence of each temporal relation defined between the next symbolic time interval and the current extended TIRP's time intervals is verified.

The boolean method (predicate) NoInst? searches for an instance of a pair of symbolic time intervals having the given temporal relation; thus, it gets *sym*, *rel*, and *next_sym* as indices to its fourth argument, the appropriate T^2 array entry, in which it queries the HashMap for the pair based on the entity id of the new instance ($inst^{new}$.e_id) and the first symbolic time interval instance ($inst^{new}$.sti[c.size-2-i]).

Once the HashMap returns a potential pair of symbolic time intervals, it compares them to the expected symbolic time interval—*next_sti*. If it was *not* found, the predicate *NoInst?* returns *True*, meaning that no instances of the requested pair were found for the same entity ID, and the search is immediately stopped, since at least one required temporal relation does not hold between the next instance and the instances of the TIRP to be extended (line 13); otherwise, it returns *False*, and the For loop continues to the next check. At the end of the search, if there was no break (i.e., NoInst? was never true - all temporal relations were verified), $inst^{new}$ is added to the supporting instances list, *c.inst.* This process is repeated for each of the *next_sti*. The searched instance *inst* is removed from *c.inst*, since it was replaced by all of the $inst^{new}$ that were found, and if not, it should be removed too (line 17). Finally the candidate *c* is returned with its list of supporting instances.

Note that retrieval of an indexed pair from T^2 to check that it indeed exists and is frequent, is performed in O(1) (in line 12), using the two-dimensional array created in the Karma phase, and, as indices, the two symbols and the temporal relation between them.

3.7 Reducing TIRP instances variance via pre-clustering

The definition of a non-ambiguous TIRP (see Definition 5) is *non-metric*: it pays no attention neither to the duration of the symbolic time intervals, nor to the duration of temporal gaps between them, or to overlaps or containment periods, as part of the definitions of their temporal relations; all of these aspects might significantly affect the meaning of a TIRP. This omission occurs since a symbol (which is, in fact, a predicate, such as "*Low blood pressure, as defined in the context of pregnancy*" or "*a Very high number of open connection sockets*") that holds over a symbolic time interval means the same for the purpose of the TIRP enumeration process, regardless of its duration: it is generated by the temporal-abstraction process and has identical semantics over all intervals of the same type.

This lack of a metric definition regarding the duration of the intervals that compose the TIRP, and the precise quantitative nature of each temporal relation, might pose a problem. The instances of discovered TIRPs may in reality vary significantly in the durations of the time intervals that they are composed from, or in the nature of their internal temporal relations, while still satisfying the same TIRP definition. Figure 9 presents three very different instances of the same TIRP definition, which in real world can reflect quite qualitatively different scenarios.

Therefore, to make the instances satisfying the definition of the same TIRP more homogeneous, with respect to the duration of the symbolic time intervals, we propose to first cluster the time interval instances of each symbol according to their durations. Thus, having for example a database with the interval-based symbols A, B, and C, we can first cluster their instances into k clusters according to their duration prior to the mining process by using an

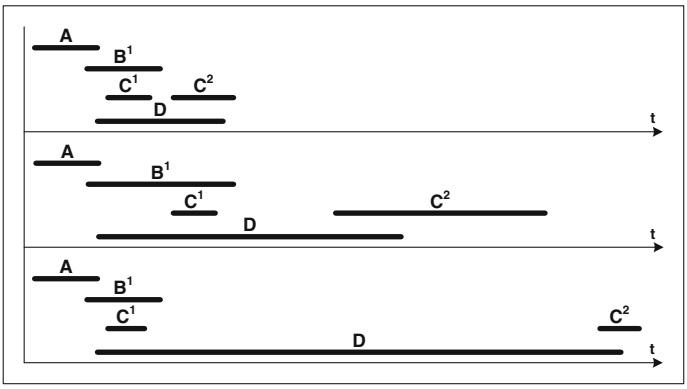

Fig. 9 Various instances of TIRPs, all satisfying the same TIRP definition, demonstrating the significant variability implied by the current TIRP definition. Note, for example, the highly different instantiations of the before relation between C^1 and C^2 and of the duration of intervals such as C^2 and D

appropriate method such as the k-means clustering algorithm. The result will be that we will have potentially more symbols, up to a maximum of k subtypes of each original symbol, and a minimum of 0 subtypes of the original symbol—in the case that none of the clusters were sufficiently frequent.

As can be readily understood, there is a trade-off involved here: pre-clustering symbolic time intervals of a specific symbol into sub-symbols according to their duration results in more types of symbols, each of which is less frequent, which results in a potentially smaller number of frequent TIRPs; but those TIRPS that are still sufficiently frequent, will be supported by more homogenous instances. With respect to the possibility of clustering also temporal *relations*, we consider it as probably unnecessary following the clustering of the time interval durations (see the discussion of this point in Sect. 5).

4 Evaluation

The objectives of the experimental evaluation were twofold. The first objective focused on a comparison of KarmaLego's performance with that of previous methods using Allen's temporal relations, on various datasets. The second objective of the evaluation focused on the assessment of the effect of pre-clustering the symbolic time intervals by their duration, prior to the mining process, on the variance of the supporting instances of the TIRPs (see Definition 9) and on their vertical support.

4.1 Datasets

4.1.1 The American signup language dataset

The American signup language (ASL) dataset was created by the National Center for Sign Language and Gesture Resources at Boston University [20]. It consists of a collection of 884 utterances, in which each utterance associates a segment of video with a detailed transcription. Every utterance contains a number of ASL gestures and grammatical fields (e.g., eyebrow raised, head tilted forward), each one occurring over a time interval. This dataset was used by [20] for the runtime evaluation of their algorithm. We used this dataset for the

runtime evaluation of our algorithm, by comparing its running time with that of the other algorithms.

4.1.2 The MavLab smart house dataset

The MavLab SmartHome dataset, provided by [10], contains data from the readings of 99 sensors installed in a computerized apartment. The sensors described the activity of people and of various appliances scattered around the apartment and were sampled each single second over a period of 81 days, in which only the data for the first 3 h from each day were used. We used this dataset for the runtime evaluation of our algorithm, by comparing its running time with the other algorithms, and also for the clustering experiments.

4.1.3 The diabetes dataset

The diabetes dataset, provided as part of collaboration with Clalit Health Services (Israel's largest HMO), contains data on 2038 patients who have type II diabetes, collected monthly from 2002 to 2007. The dataset contains six variables recorded over time for each patient: hemoglobin-A1c values, blood glucose levels, cholesterol values, and several medications the patients purchased: diabetic (insulin-based) medications, cholesterol reducing statins, and beta blockers. The total amount of the diabetic medications was represented in the dataset in terms of an overall defined daily dose (DDD).

Knowledge-based state-abstraction definitions for abstraction of the raw data laboratory-test measurements into more meaningful concepts were provided by expert physicians from Ben Gurion University's Soroka Medical Center, as shown in (Table 3 in Appendix 6).

4.2 Example of knowledge discovery using the KarmaLego algorithm

To make our performance analysis and the overall motivation for making the mining process more efficient and more concrete, Figure 10 shows a typical result of discovered TIRPs, in this case, a case study we performed in the diabetes domain [18]. For example, 141 of the 1,702 patients who had an Increasing period of the HbA1C measurement (HBA1C.inc) [denoting an exacerbation of their disease], finished that period by an interval of an Increasing dose of the diabetes medications (Diab.inc), followed immediately by (i.e., *meeting*) an interval

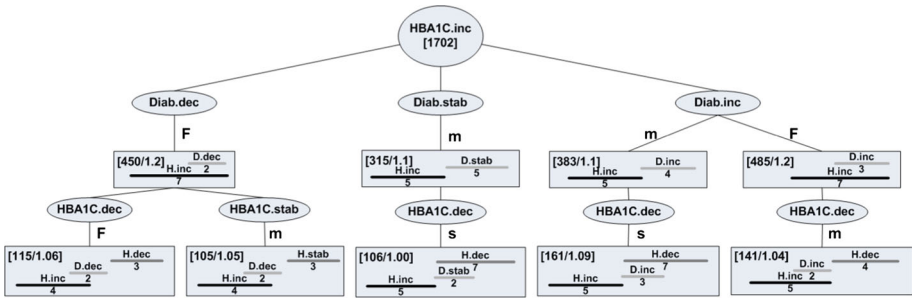

Fig. 10 A part of the TIRP tree in the diabetes domain. For each node, the level of [vertical] support is given; within each node are shown the number of entities found to have at least one instance of the TIRP (i.e., the vertical support) and the mean number of instances of that TIRP found in each entity, displayed as [vertical support/mean number of instances per entity] D.inc, D.dec, D.stab = drug dose gradient abstractions; H.dec, H.inc, H.stab = HbA_1C gradient abstractions; F = Finishes; M = Meets; S = Starts (temporal relations). The time intervals within each node denote the mean duration of the symbolic intervals of each node, in months.

of Decreasing HbA1C levels (HBA1C.dec) (denoting an improvement in the disease). Other patterns of behavior can be found as well.

Another useful way to characterize a cluster of patients by their temporal pattern is by the distribution of the static, non-temporal, variables characterizing these patients, which were *not* part of the TIRP-discovery process (e.g., gender, age group). For example, during our evaluations, in the case of the diabetes data set, it seemed that certain TIRPs, such as those describing the response of the HbA1C parameter (which approximates the severity of the diabetic state) to treatment by diabetes medications, are significantly more frequent in male rather than in female patients. Such insights are easy to arrive at and explore further, once a TIRP tree exists.

4.3 A runtime evaluation of KarmaLego on the datasets

We evaluated the hypothesis that the KarmaLego algorithm improves runtime performance for discovery of TIRPs. Thus, a comparison of the runtimes was performed among (1) the KarmaLego method, (2) an implementation of the H-DFS method [20], (3) an implementation of ARMADA [28] and (4) an implementation of IEMiner [21] method. All the methods were implemented in Visual C++ according to their papers. *We made sure that all the methods produce the same output of frequent patterns.*

Since several of the methods detect and extend only the first instance of a TIRP, and not all of the instances of its occurrence within each entity (unlike the KarmaLego method), all the TIRP-discovery methods were applied that way for the evaluation. All experiments were conducted on a 3 Ghz Intel Xeon server, having 3Gb main memory, running Microsoft WindowsServer2003. We used the *ASL*, *Diabetes* and *SmartHome* data sets described above.

Table 1 describes the data sets using four parameters: N: the entire number of time intervals in the dataset; E: the number of entities (i.e., patients or subjects); S: the number of different symbols; and I: the mean number of intervals per entity. While the ASL dataset is relatively small in all the parameters, the Diabetes dataset contains a significantly larger number of time intervals (N) and of entities (E). The SmartHome dataset has the largest mean number of time intervals per entity (I).

All four methods, including: H-DFS, Armada, IEMiner, and KarmaLego, were run on the ASL dataset using 5 levels of *minimal vertical support*, starting with 50 % and ending with 10 %. Figure 11 presents the runtime in seconds, as a function of the minimal vertical support (%), using a logarithmic scale for the computation time.

In this dataset, Armada and KarmaLego behaved very similarly, while IEMiner was slower, and the H-DFS method was much slower (see Fig. 11). The similar behavior of Armada and KarmaLego on this data set can be explained by the relatively low value of I, the mean number of time intervals per entity, in this case a subject, which enables the Armada method to keep up with the KarmaLego method. As will be shown later, this is not the case when the mean number of intervals per entity is large, causing more difficulties to the Armada method. Since the runtime of the H-DFS method is significantly slower than that of the other methods,

Table 1 The properties of the evaluation datasets

N number of time intervals, S number of symbols, E number of entities, I mean number of intervals per entity

Dataset	N	E	S	I
ASL	2,037	65	146	31.3
Diabetes	**80,538**	**2,038**	227	39.5
SmartHome	30,210	81	212	**372.9**

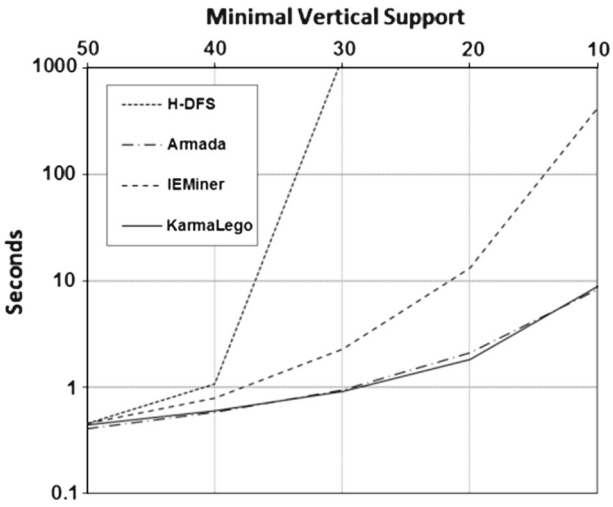

Fig. 11 A runtime comparison of the algorithms (in seconds) on the ASL data set as a function of the minimal vertical support threshold (%). *Note* time is presented using a logarithmic scale.

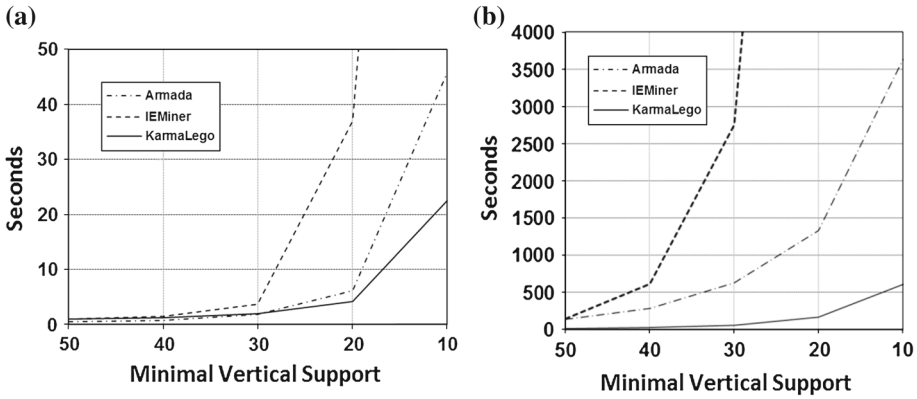

Fig. 12 A performance comparison among three TIRP mining algorithms applied to the Diabetes (**a**) and SmartHome (**b**) data sets. Computation time (in seconds) is displayed as a function of the minimal vertical support threshold (in percentages). The KarmaLego algorithm is faster than both others, especially for low levels of vertical support

it requires a logarithmic-scale graph for the clarity of presentation. Thus, its performance results are presented in this evaluation only once, using the relatively small ASL dataset.

Figure 12a shows the results of the experiments on the Diabetes dataset, in which N and E are the largest. While IEMiner was the slowest, KarmaLego was about twice as fast as Armada at very low levels of minimal vertical support. Experiments on the SmartHome dataset, having the largest mean number of intervals per entity, I, are shown in Fig. 12b. KarmaLego was much faster than both IEMiner and Armada. The difference was pronounced more clearly, compared with the ASL and Diabetes data sets, due to the large size of the SmartHome data set.

4.4 Experiments in pre-clustering time intervals by their duration

Our second hypothesis in the current study was that pre-clustering the symbolic time interval instances in the dataset by their duration reduces not only the number of discovered TIRPs

when the clustering parameter k (number of clusters) increases, but also the mean time interval variance (TIV) for discovered TIRPs (definition 9), due to the increased uniformity, at least in symbolic time interval duration, of the components of the discovered TIRPs.

In these experiments, which focused on TIRP counts and on interval *duration* variance measures, we used an epsilon value of zero (the Allen default); three values of k for the K-means clustering: two, three and four; and the non-clustered data for comparison. To evaluate the influence of these settings on the variance of the TIRPs time intervals instances variance, we counted the number of TIRPs discovered, C, and measured the TIV of discovered TIRPs.

The Diabetes dataset was abstracted into either 3 or 9 states using the *Equal-Width Discretization* (*EWD*) method (see Sect. 2.2), creating two datasets: Diabetes_3EWD and Diabetes_9EWD, respectively. The MavLab SmartHome dataset was abstracted into 3 states using EWD. The symbolic intervals of both datasets were pre-clustered by duration using the K-means method into $k = 2$, 3 and 4 clusters. The number of symbolic interval types in the dataset after each size of k pre-clustering is presented in Table 1. Note that the number of symbolic interval types after pre-clustering is not necessarily multiplied (approximately) by the k value, but it is bounded by that value, since the clusters may sometimes be empty.

Table 2 presents the quantitative properties of each dataset. Note that the number of symbolic time intervals after a 9EWD pre-clustering process is approximately twice than after a 3EWD state abstraction, since there are now many additional symbolic interval types—each symbol type is fragmented into several subtypes (states), by its value range; thus, the same is true for the mean number of symbolic intervals per entity. Furthermore, the number of symbolic interval types increases as the number of clusters grows, due to a similar fragmentation process, this time by duration.

Figures 13 and 14 display an analysis of the pre-clustering results for the Diabetes_3EWD dataset, for the 3-sized and 4-sized TIRPs. In both figures, the TIV decreases when the number of clusters increases. Thus, the discovered TIRPs are indeed more homogenous with respect to their duration. However, as expected, the *number* of discovered TIRPs also decreases, for the reasons explained in Sect. 3.7. Note that the number of discovered TIRPs with four clusters was zero, in the case of the 4-sized TIRPs. (Similar trends were noted after using 9EWD discretization.)

Figures 15 and 16 present the analysis results for the SmartHome_3EWD dataset, for discovery of 3-sized and 4-sized TIRPs, respectively. In both figures, the TIV decreases significantly as the number of clusters increases, while the number of frequent TIRPs decreases. While the TIV decreased significantly for both 3-sized TIRPs (Fig. 15) and 4-sized TIRPs (Fig. 16), the number of discovered frequent TIRPs decreased less dramatically, especially for 4-sized TIRPs. Unlike in the case of the Diabetes datasets, the 4-sized TIRPs were frequent

Table 2 The evaluation datasets properties after the pre-clustering

Dataset	N	E	I	NC	C2	C3	C4
D_3EWD	**36,434**	**2,038**	17.8	101	150	192	226
D_9EWD	**73,144**	**2,038**	35.8	234	336	410	463
SH_3EWD	42,847	88	**486.9**	232	366	477	573

N = no. of symbolic time intervals; E = no. of entities; I = mean no. of symbolic intervals per entity; NC = no. of symbolic interval types with no clustering; $C2$ = no. of symbolic intervals types for $k = 2$ clusters; $C3$ = no. of symbolic intervals types for $k = 3$ clusters D diabetes, SH SmartHome

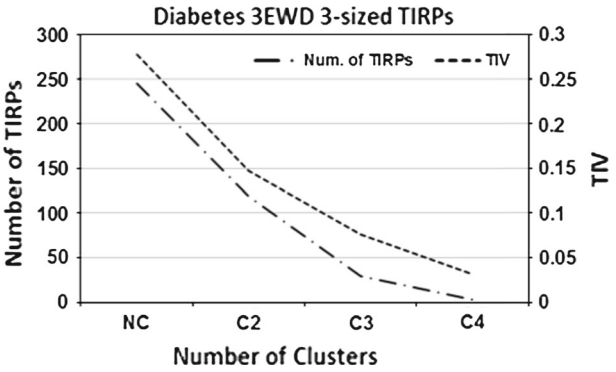

Fig. 13 The *time-intervals-variance* (TIV) decreases as the number of clusters grows, in the case of 3-sized TIRPs in the Diabetes dataset. A similar observation applies to the *number* of discovered 3-sized frequent TIRPs

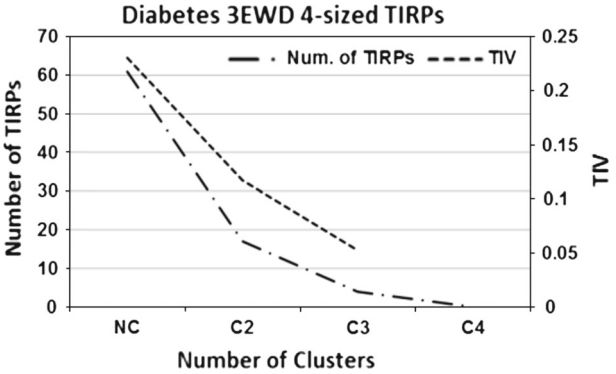

Fig. 14 The *time-intervals-variance* (TIV) decreases as the number of clusters grows, in the case of 4-sized TIRPs in the Diabetes dataset. A similar observation applies to the *number* of discovered 4-sized frequent TIRPs

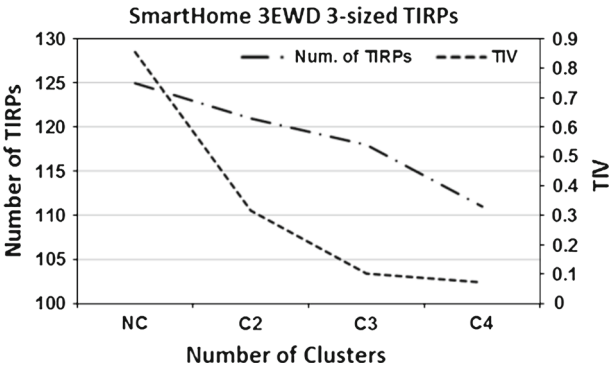

Fig. 15 The time-intervals-variance (TIV) decreases as the number of clusters grows, for 3-sized TIRPs in the SmartHome dataset, as does the *number* of discovered 3-sized frequent TIRPs

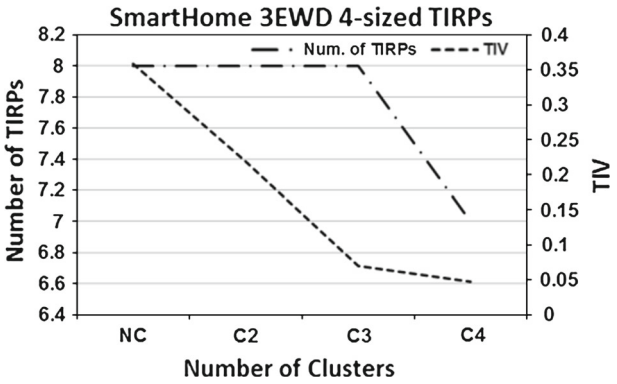

Fig. 16 The time-intervals-variance (TIV) decreases as the number of clusters grows, for 4-sized TIRPs in the SmartHome; the *number* of frequent TIRPs decreases only with 4 clusters

even with clustering by duration into three and four clusters. This can be explained by the larger number of time intervals per entity in that dataset (Table 1).

5 Summary and discussion

5.1 A summary of the main results and the conclusions from them

We presented a comprehensive process based on a new algorithm, KarmaLego, for efficient temporal knowledge discovery from multivariate temporal data. The overall process includes a TA preprocessing phase of the raw time-stamped data into meaningful symbolic time intervals.

Note that the TA preprocessing phase overcomes several frequent problems encountered in temporal data mining, since temporal multivariate data often appear in varying granularities and frequencies, and some of the raw data might be missing; TA (which typically includes some form of interpolation) often solves such problems.

The overall temporal data mining process uses an extended version of all of Allen's temporal relations, relying on constrained-based representation to define in an internally coherent, robust fashion all of the temporal relations by an epsilon value. The result is a mutually exclusive flexible definition of all of Allen's seven basic temporal relations. Note that the KarmaLego algorithm handles *any* number of temporal relations used during the enumeration process, and is oblivious to the degree of fuzziness (i.e., the epsilon value) used in their definition.

We have also introduced several fundamental concepts that define the output of the TIRP mining process, such as the vertical support and the Time Interval Variance. We then explained in detail the fast time intervals mining technique used within the KarmaLego algorithm, which extends TIRPs directly using a specialized data structure, and which exploits the transitivity of the temporal relations to significantly reduce the number of generated candidates.

We demonstrated the use of the KarmaLego algorithm on several datasets, and in particular, on a set of diabetic patients' records; we have presented an example of a part of the discovered TIRPs tree, in the case of the diabetes domain, whose patterns can be quite useful to clinical researchers.

The quantitative performance evaluation first compared the runtime performance of KarmaLego with other alternative methods: ARMADA, IEMiner, and H-DFS. Although the ARMADA algorithm is relatively fast, it becomes very slow, in large datasets, especially when the number of time intervals per entity is large, such as in the SmartHome dataset, and when using low levels of minimal vertical support. This can be explained by the fact that the search in ARMADA is performed on the actual data, without indexing first the 2-sized TIRPs. While in both KarmaLego and the IEMiner method, the construction of TIRPs is performed on pairs of time intervals that are indexed earlier (the Karma algorithm within KarmaLego), the IEMiner method is much slower than KarmaLego, since in KarmaLego the transitivity-based candidate generation phase makes the candidate generation process significantly more efficient and faster, resulting in a significantly smaller number of candidates to examine.

Finally, to provide a better basis for our TIRP-discovery framework, we considered the common problem of a lack of metric specification regarding the time intervals durations in the otherwise non-ambiguous definition of TIRPs and the problem of discovering TIRPs that significantly vary with respect to the duration of their supporting instances. For that purpose, we proposed to pre-cluster the symbolic time intervals by their durations into k clusters.

We demonstrated the feasibility of this process and quantitatively assessed its effects and its potential trade-offs using several real datasets. The use of pre-clustering indeed increased the homogeneity of the instances, as witnessed by a lower TIV measure, as we hypothesized, but, as we suspected, has resulted also in the reduction in the number of discovered frequent TIRPs.

In addition to clustering intervals by *duration* and measuring the variance in the duration of the TIRP's interval-based components, it is also possible in theory to cluster temporal *relations* by their characteristics, such as by the duration of the gap between the two intervals in the case of the *before* relation, or the duration of the overlapping segment, in the case of the *Overlaps* relation.

However, pre-clustering time intervals by their *duration* greatly decreases the need for clustering the temporal relations by other characteristics, such as clustering the relations *starts*, *finished-by*, or *equal*. For example, the quantitative characteristics of the temporal relation between two time intervals, when both of the start-time points, or both of the end-time points, are identical (as is the case for the *starts or finished-by* relations), or both the start-time and the end-time points are identical (as is the case with *equal*), are already considerably constrained by the duration-based pre-clustering and thus have by nature already very low variability. For example, the *starts* sub-relations can include only up to the small number of combinations of intervals from the discovered classes of the two relevant symbols (e.g., $2 \times 3 = 6$, when 2 classes were discovered for one symbol and 3 classes for the other, after clustering by duration) while the *equal* sub-relation can include only up to the lower number of classes of each symbolic interval that were discovered during the clustering by duration (e.g., 3 sub-relations, when 3 duration classes were discovered for one symbol and 4 duration classes were discovered for the other symbol).

We expect that following the time interval duration-based pre-clustering, only the clustering of the gap durations in the case of the *before* relation, and possibly the relative duration of the overlapped segment in the case of the *overlap* relation, and in the case of the *contain* relation, might be meaningful. (Note that we are already using a *maximal gap* parameter, in the case of the *before* temporal relation, thus considerably reducing the potential variability).

The main downside of pre-clustering temporal relations into sub-relations (as happens with the time intervals pre-clustering), in addition to clustering by durations, is that it reduces the support for each such sub-relation to a level below the minimal threshold.

5.2 The implications of an effective TIRP-discovery methodology

The effective discovery of temporal patterns in the data, referred to as TIRPs in this paper, supports temporal knowledge discovery, as well as potentially facilitating clustering of the entities by their temporal behavior, and classification of multivariate time series, including a form of prediction.

Indeed, we have previously shown that using TIRPs from time-stamped data collected during the first 12 h of hospitalization in an intensive-care unit (ICU), as features for prediction whether an ICU patient will require ventilation for more than 24 h, resulted in a positive predictive value of 79 %, given a population of 664 patients, of which 28 % needed more than 24 h of ventilation [19].

Each TIRP can also be viewed as representing a *cluster* of entities, such as patients, who have similar temporal behavior along time, such as for example, a similar pattern of a reaction to a particular drug dose. Each cluster, or TIRP P, can be described by its vertical support, the mean number of instances of the TIRP found within each entity, and other measures such as TIV. The extended (longer) TIRPs have equal or smaller vertical support; these are actually sub-cluster variations within a k-sized TIRP population. Thus, in the case of the medical domain, the same chronic disease might express itself in a population of patients through only a certain set of meaningful temporal pathways, i.e., possible courses of disease. Indeed, that is what we discovered in the case of the diabetes domain [18].

Acknowledgments The authors wish to thank Panagiotis Papapetrou for sharing datasets and insightful discussions, as well as Christos Faloutsos, Christian Freksa, Frank Hoppner, Fabian Moerchen and Dhaval Patel for insightful discussions about time intervals mining. This work was supported in part by grants from the Deutsche Telekom Laboratories at Ben Gurion University, and by the HP labs Innovation Research Program, Award No. 2008-1023.

6 Appendix

Table 3 contains the cutoff definitions used for each state in the case of the raw measurements included in the Diabetes dataset. The rest of the raw data consisted of medications, for each of which an overall defined daily dose (DDD) abstraction was defined.

The knowledge-based state-abstraction definitions for the measurements were provided by physicians from Ben Gurion University's Soroka Medical Center.

In the following four parts of Table 3, the cutoff definitions are presented for each state for the various temporal measurement variables in the Diabetes dataset.

Table 3 Measurement data types and their cut-off (discretization) values for 3 or 4 state abstractions each, for four concepts, in the case of the diabetes data set

State	Blood glucose	HemoglobinA1C	LDL cholesterol	HDL cholesterol	
				HDL-male	HDL-female
1	<100	<7	<100	<35	<30
2	100–125	7–9	100–130	35–45	30–40
3	126–200	9–10.5	130–160	>45	>40
4	>200	>10.5	>160		

References

1. Allen JF (1983) Maintaining knowledge about temporal intervals. Commun ACM 26(11):832–843
2. Ayres J, Gehrke J, Yiu T, Flannick J (2002) Sequential pattern mining using bitmaps. In: Proceedings SIGKDD international conference on knowledge discovery and data mining. Edmonton, Alberta
3. Azulay R, Moskovitch R, Stopel D, Verduijn M, de Jonge E, Shahar Y (2007) Temporal discretization of medical time series: a comparative study. IDAMAP, Amsterdam
4. Batal I, Sacchi L, Bellazzi R, Hauskrecht M (2009) A temporal abstraction framework for classifying clinical temporal data. American Medical Informatics Association (AMIA) Annual Symposium Proceedings, pp 29–33
5. Bellazzia R, Diomidousb M, Takabayashid K, Zieglere A, McCray AT (2011) Data analysis and data mining: current issues in biomedical informatics. Methods Inf Med 50(6):536–544
6. Freksa C (1992) Temporal reasoning based on semi-intervals. Artif Intell 54(1):199–227
7. Fu TK (2011) A review on time series data mining. Eng Appl Artif Intell 24:164–181
8. Höppner F (2001) Learning temporal rules from state sequences. In: Proceedings of WLTSD-01
9. Höppner F (2002) Time series abstraction methods: a survey. In: Workshop on knowledge discovery in databases. Dortmund
10. Jakkula VR, Cook DJ (2011) Detecting anomalous sensor events in smart home data for enhancing the living experience. Artificial intelligence and smarter living, WS-11-07 of AAAI Workshops, AAAI
11. Kam PS, Fu AWC (2000) Discovering temporal patterns for interval based events. In: Proceedings DaWaK-00
12. Lavrač N, Keravnou-Papailiou E, Zupan B (1997) Intelligent data analysis in medicine and pharmacology. The Springer international series in engineering and computer science, vol 414. Kluwer Academic Publishers, Boston/Dordrecht/London
13. Lin J, Keogh E, Lonardi S, Chiu B (2003) A symbolic representation of time series with implications for streaming algorithms. In: ACM SIGMOD DMKD workshop
14. Mörchen F, Ultsch A (2005) Optimizing time series discretization for knowledge discovery. In: Proceeding of KDD
15. Mörchen F (2006) Algorithms for time series knowledge mining. In: Proceedings of KDD
16. Moskovitch R, Stopel D, Verduijn M, Peek N, de Jonge E, Shahar Y (2007) Analysis of ICU patients using the time series knowledge mining method. IDAMAP, Amsterdam
17. Moskovitch R, Elovici Y, Rokach L (2008) Detection of unknown computer worms based on behavioral classification of the host. Comput Stat Data Anal 52(9):4544–4566
18. Moskovitch R, Shahar Y (2009) Medical temporal-knowledge discovery via temporal abstraction. AMIA, San Francisco
19. Moskovitch R, Peek N, Shahar Y (2009) Classification of ICU patients via temporal abstraction and temporal patterns mining, IDAMAP 2009. Verona
20. Papapetrou P, Kollios G, Sclaroff S, Gunopulos D (2009) Mining frequent arrangements of temporal intervals. Knowl Inf Syst 21(2):133–171
21. Patel D, Hsu W, Lee ML (2008) Mining relationships among interval-based events for classification, In: Proceedings of the 2008 ACM SIGMOD international conference on management of data, pp 393–404
22. Pei J, Han J, Mortazavi-Asl B, Pinto H, Chen Q, Dayal U, Hsu M-C (2001) PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceedings of the 17th international conference on data engineering. pp 215–224
23. Roddick J, Spiliopoulou M (2002) A survey of temporal knowledge discovery paradigms and methods. IEEE Trans Knowl Data Eng 4(14):750–767
24. Sacchi L, Larizza C, Combi C, Bellazi R (2007) Data mining with temporal abstractions: learning rules from time series. Data Min Knowl Discov 15:217–247
25. Shahar Y (1997) A framework for knowledge-based temporal abstraction. Artif Intell 90(1–2):79–133
26. Shahar Y (1999) Knowledge-based temporal interpolation. J Exp Theor Artif Intell 11:123–144
27. Villafane R, Hua K, Tran D, Maulik B (2000) Knowledge discovery from time series of interval events. J Intell Inf Syst 15(1):71–89
28. Winarko E, Roddick J (2007) Armada: an algorithm for discovering richer relative temporal association rules from interval-based data. Data Knowl Eng 1(63):76–90
29. Wu S, Chen Y (2007) Mining non ambiguous temporal patterns for interval-based events. IEEE Trans Knowl Data Eng 19(6):742–758

Robert Moskovitch holds a B.Sc., M.Sc., and a Ph.D. in Information Systems Engineering from Ben Gurion University and is currently a postdoctoral research scientist at the Department of Biomedical Informatics at Columbia University. Prior to that, he headed several Research and Development projects in Information Security at Deutsche Telekom Innovation Laboratories at BGU. He has served on several journal editorial boards, as well as on program committees of several conferences and workshops in Biomedical Informatics and in Information Security. He published more than fifty peer reviewed papers in leading journals and conferences, several of which had won best-paper awards.

Yuval Shahar is a professor and previous chair of BGU's Information Systems Engineering department. He holds a B.Sc. and an M.D. (Hebrew University), an M.Sc. in computer science (Yale University), and a Ph.D. in Medical Information Sciences (Stanford University). After a decade at Stanford as a researcher and faculty member (Medicine and Computer Science), he joined BGU in 2000 to found and head its Medical Informatics Research Center. His research focuses on temporal reasoning, temporal data mining, therapy planning, information visualization, knowledge acquisition, knowledge representation, and decision analysis, applied to biomedicine, homeland security, and information security. He published more than 200 papers. Among multiple awards, Prof. Shahar was granted in 1995 the NIH 5-year FIRST career award; in 2005, an IBM Faculty Award, and in 2008, an HP Worldwide Innovation Program award. He was elected in 2005 as an International Fellow of the *American College of Medical Informatics* (ACMI).