R. Moskovitch, Y. Shahar. Fast Detection of Time Intervals Related Patterns. Technical Report 11/14.,Ben Gurion University, Beer Sheva, Israel, 2014.

Fast Detection of Time Intervals Related Patterns

Robert Moskovitch[1,2], Yuval Shahar[1]

[1] Department of Information Systems Engineering, Ben Gurion University, Israel.
[2]Department of Biomedical Informatics, Systems Biology, and Medicine,
Columbia University, New York, USA.
{robertmo,yshahar}bgu.ac.il

Abstract. Classification of multivariate temporal data records is an important, though challenging task. Such records include a large number of time-oriented variables sampled in irregular fashion, often represented both by time point and time intervals, thus providing several challenges for analysis and data mining. Examples include medical records, financial records, and information security log files. Increasingly, temporal abstraction, in which a series of raw-data time points is transformed into a set of symbolic time intervals, is being used as a preprocessing step to discover time intervals related patterns (TIRPs) as features for classification. One example is our own framework, KarmaLegoSification (KLS), which uses the KarmaLego algorithm to discover frequent TIRPs. However, in addition to the TIRPs discovery phase, there is the need to detect in each single entity to be classified, a large number of TIRPs, determined during the training phase to be potentially useful features, a task that is often quite time consuming. Thus, we had developed a version of the KarmaLego algorithm, which we refer to as SingleKarmaLego (SKL), to detect multiple TIRPs within a single entity's set of symbolic time intervals. In this study, after presenting briefly the KarmaLego algorithm and analyzing its worst-case behavior, we present the SKL algorithm and analyze its worst-case and average-case behavior when compared to a standard Sequential TIRPs Detection (STD) algorithm. We demonstrate the clear theoretical advantage of the SKL algorithm in both cases, in spite of its initial higher setup cost for indexing the relations amongst all of the entity's symbolic-interval pairs. Then, we evaluate the two algorithms' run times on three datasets from the quite different medical domains of diabetes, intensive care, and infectious hepatitis. The empirical results validate the theoretical analysis and demonstrate a significant speed up obtained by using the SKL algorithm versus the STD algorithm.

Keywords: Temporal Abstraction, Temporal Pattern Matching, Time Intervals Mining, Temporal Knowledge Discovery, Temporal Data Mining, Frequent Pattern Mining, Classification.

1 Introduction

Data scientists are witnessing an increasing availability of longitudinal electronic data in multiple domains. A particularly poignant case in point is the biomedical domain. This new availability presents a significant opportunity to discover new knowledge, such as new medical knowledge, from multivariate, time-oriented data, such as longitudinal clinical records, and to perform various classification tasks based on the temporal data, such as for purposes of diagnosis (e.g., a correct interpretation of a series of clinical data), plan recognition (e.g., recognizing and understanding a care provider's plan), quality assessment (e.g., comparing the course of therapy to a gold standard pattern emerging from the records of multiple other patients), and prediction of meaningful clinical outcomes.

However, temporal data in general, and in biomedical domains in particular, include not only time-stamped raw data, or time *points* (e.g., a blood-glucose value of 142 mg/deciliter, at 07:20 A.M., on February 12th, 2014)), 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 the drug Metformin for 21 days), or are *abstractions*, or interpretations, derived from them (e.g., two weeks of *high fasting blood glucose*). Processing such data requires special care.

Classification of temporal data, especially of both uni-variate and multivariate time series, is a highly challenging as well as an important task in many domains, such as information security, in which classification can be used for malware detection (Moskovitch et al, 2008), financial domains, in which behavioral classification can detect patterns of potential fraud, and many other time-oriented domains. However, it is especially essential in a multitude of different medical domains, in which correct classification of time-series data has immediate implications for diagnosis, for quality assessment, and for prediction of meaningful outcomes (Sacchi et al., 2007; Moskovitch et al, 2009; Battal et al, 2012; Hauskrecht et al, 2013). Longitudinal data, especially multivariate data, are often complex, are typically heterogeneous in type and in format, and are usually measured in irregular time periods.

Several studies have suggested that it is worthwhile to transform the time-point series, such as individual Hemoglobin measurements, into a series of symbolic time intervals, such as periods of Anemia at various levels, a process that we refer to as *temporal abstraction*. This preprocessing overcomes much of the problems of varying-frequency measurements and recordings, and of minor measurement errors and deviations in the raw data, through the creation of concepts that are no longer time-stamped, raw data, but rather interval-based abstractions, or interpretations, of these data, and through the smoothing effect of these abstractions. Then, one can mine these time-interval series for frequent time-intervals-related-patterns (TIRPs), and induce a classifier that exploits these frequent TIRPs as features. To classify a new entity, we can first abstract its raw data into a set of symbolic time intervals, and then detect the set of TIRP features within the single entity, to classify it according to the induced classifier. We can thus answer a classification question, such as, "Does this hepatitis patient suffer from Hepatitis A or from Hepatitis B?" or a prediction question, such as "What is the likelihood of this diabetes patient to develop future renal damage?"

Indeed, we have developed a classification framework, the *KarmaLegoSification* (KLS) framework, which embodies this overall temporal classification process, and in particular, the use of TIRPs as classification or prediction features. The KLS framework exploits a highly efficient algorithm, KarmaLego (Moskovitch and Shahar, In Press), which we briefly present in Section 3, for the discovery of frequent TIRPs within a set of entities, each represented as a series of symbolic intervals.

The use of frequent symbolic time intervals, i.e., TIRPs, as features for the classification of multivariate time series, was proposed first in (Patel et al, 2008). Since these early studies, several additional studies were published exploring the use of frequent TIRPs as classification features for multivariate time series classification [Sacchi et al., 2007; Moskovitch et al, 2009; Patel et al, 2008; Batal et al, 21012a; Batal et al, 2012b; Hauskrecht et al, 2013; Moskovitch and Shahar, 2013]. The various studies using TIRPs for classification highlight the need for efficient detection in single entities of the multiple TIRPs that have been determined to be useful features for classification purposes.

Note that the TIRPs detection problem is common to any task that attempts to detect multiple TIRPs within a set of individual entities. That includes all of the frameworks that use TIRPs as classification features, as well as any task involving TIRP-like queries to an entity database.

However, detecting the existence of a set of TIRPs (when these TIRPs serve as features, or for any other purpose) in an entity or a set of entities might require significant effort, especially if there are many TIRPs to be detected, and if there are many symbolic time intervals within each entity, and even more so, if some of the TIRPs are rather long. **The efficient detection of these TIRPs within a set of entities, each represented as a set of symbolic time intervals, is the focus of the current paper**.

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

1. Introducing *a novel algorithm for detection of multiple TIRPs within one or more single entities* represented as a series of symbolic intervals, i.e., the *SingleKarmaLego* (SKL) algorithm;
2. *Analyzing the complexity of the SKL algorithm,* comparing both its worst-case behavior and average-case behavior to those of a standard *Sequential TIRPs Detection* (STD) algorithm, and showing the significant theoretical superiority of the SKL algorithm in both cases, in spite of its relatively high initial setup cost;
3. Performing a set of experiments comparing the runtime behavior of the SKL algorithm to that of STD algorithm on three different sets of longitudinal, multivariate clinical data, demonstrating empirically the significant speed-up in run time provided by the SKL algorithm, compared to the STD algorithm's runtime.

The rest of this paper is organized as follows: We start by introducing briefly, in the Background (Section 2), the concepts of temporal data mining, temporal abstraction, temporal discretization, time-interval pattern mining, and classification based on patterns – specifically, based on time-interval patterns. We then briefly introduce in Section 3 a fast time-intervals mining method that we had previously developed, called KarmaLego, and show how it can be used to discover frequent TIRPs. We then analyze the worst-case complexity of TIRP detection when using a KarmaLego-like method. In Section 4, we introduce the *SingleKarmaLego* (SKL) algorithm for detection of multiple TIRPS within one or more single entities represented as a series of symbolic intervals. Then, we first compare the worst-case complexity of SKL to that of a standard Sequential TIRPs Detection algorithm, and then follow by comparing the behavior of both algorithms when presented with an "average" case of TIRP detection. In Section 5, we demonstrate the validity of the theoretical analysis by comparing the run time behavior of the SKL to that of the

Sequential TIRPs Detection algorithm, by applying both to three different longitudinal, multivariate clinical data sets. We summarize our main contributions and discuss their implications in Section 7.

2 Background

2.1 Temporal Data Mining

Temporal data mining 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 amongst 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 embodies within it a considerable promise for the understanding of various scientific phenomena, and the potential for creation of richer and more accurate classification models, representing explicitly processes developing over a long time. An excellent survey of temporal data mining can be found in (Roddick and Spiliopoulou, 2002). Enhancing the use of the temporal dimension as part of the data mining process is an emerging need in the biomedical domain (Bellazi et al, 2011).

2.2 Temporal Abstraction and Discretization

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 raw Hemoglobin-value or liver-enzyme measurements, more abstract characterizations such as "3 weeks of moderate anemia", or "5 months of decreasing liver functions"), suitable for human inspection or for data mining.

TA typically includes some process for transforming raw, numeric data into a set of symbolic values, a process known as *discretization*. In particular, *Temporal discretization* refers to the process of discretization of a time-series values, as a preprocessing step in transforming the time-stamped, raw-concept series into a set of symbolic, state-based time intervals. Common discretization methods include *Equal Width Discretization (EWD),* which uniformly divides the ranges of each value, and *Equal Frequency Discretization* (*EFD*), do not consider the temporal order of the values; other methods, such as *Symbolic Aggregate approXimation* (*SAX*) (Lin et al, 2003) (which focuses on a statistical discretization of the values) and *Persist* (Moerchen and Ultsch, 2005) (which maximizes the duration of the resulting time intervals), explicitly consider the temporal dimension. In previous studies, we have compared several versions of these methods, especially for the purpose of discretization of time-oriented clinical data (Azulay et al, 2007) and eventually for the purpose of classification (Moskovitch and Shahar, 2009).

A TA process usually includes also some form of *interpolation* among the abstracted point-based states, to form intervals, and among the intervals, to generate longer-duration intervals (Shahar, 1999).

TA 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. TA for time series mining in the form of time intervals was already proposed by Hoeppner (2002). 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 1 shows a TA process for one concept.

There are several approaches to the TA task; some of these exploit context-sensitive knowledge acquired from human experts, a method known as *knowledge-based temporal abstraction* (KBTA) (Shahar, 1997); others are purely automatic, and rely mostly on a discretization of the raw values and concatenation (Azulay et al, 2007; Hoeppner, 2002; Moerchen and Ultsch, 2005).

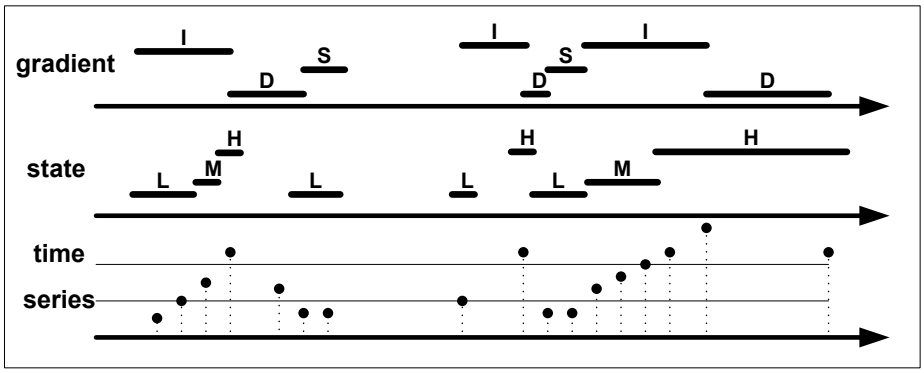

Figure 1: A series of raw time-stamped data of one concept type (at the bottom) is abstracted into an interval-based *state* abstraction (i.e., a value classification) that has, in this particular case, three discrete values: Low (L), Medium (M), and High (H) (in the middle); and into a *gradient* abstraction (i.e., the sign of the first derivative) that has the values Increasing (I), Decreasing (D), and Stable (S) (at the top).

2.3 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 (Villafane et al, 2000). One of the earliest studies in the area is that of Villafane et al. (2000), which searches for containments of time intervals in a multivariate symbolic time interval series. Kam and Fu (2000) were the first to use all of Allen's temporal relations to compose time interval series, but their patterns were ambiguous, since the temporal relations among the components of a pattern are undefined, except for the temporal relations among all of the pairs of successive intervals.

Hoeppner (2001) 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 pair-wise relations within a k-intervals pattern. 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 Section 3 (Definition 5). Papapetrou et al. (2009) propose a hybrid approach H-DFS, which combines the first indexing the pairs of time intervals and then mining the extended TIRPs in a candidate generation fashion. 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, and introduced an *epsilon threshold*, to make the temporal relations more flexible.

ARMADA, by Winarko and Roddick (2007), is a relatively recent projection-based efficient time interval mining algorithm that uses a candidate generation and mining iterative approach. Wu et al. (2007) proposed TPrefixSpan, which is a modification of the PrefixSpan sequential mining algorithm (Pei et al, 2001) for mining non-ambiguous temporal patterns from interval based events. Patel et al. (2008) introduced IEMiner - a method inspired by Papapetrou's method, which extends the patterns directly, unlike Papapetrou et al.'s method; direct extension is in fact performed also in the KarmaLego method (see Section 3.2). Patel et al. (2008) had compared their method runtime to TPrefixSpan (Wu et al, 2007) and H-DFS (Papapetrou et al, 2009) and found their method to be faster.

Moskovitch and Shahar (2009; In Press) introduced the KarmaLego algorithm, which performs fast time interval mining by extending TIRPs directly and by exploiting the transitivity of temporal relations to generate candidates efficiently; the KarmaLego methodology is at the basis of the KLS classification framework presented here, which motivates our examination of the benefits of the SKL algorithm. (We present the KarmaLego algorithm briefly in Section 3).

Other methods for time intervals mining were proposed, which either do not use Allen's temporal relations (Moerchen, 2006), or use only a subset of these relations, abstracted into a super-relation, such as *Precedes*, which is the disjunction of *Before*, *Meets*, *Overlaps*, *Equal*, *Starts*, and *Finished-By*, and which can form a basis for the discovery of a set of temporal association rules (Sacchi et al, 2007).

2.4 Classification via Frequent Patterns

The increased attention to the subject of mining time intervals has led several research groups to simultaneously propose using the discovered temporal patterns as features for classifying multivariate time series (Batal et al, 2012; Patel et al, 2008), including a preliminary version of our current classification framework (Moskovitch et al, 2009). Interestingly, all of the studies that reported the use of temporal abstraction and time intervals mining for the purpose of classification were using datasets from the biomedical domain (Patel et al, 2008; Batal, 2012).

Patel el. at (2008) presented a time intervals mining algorithm, called IEMiner and used the discovered patterns for classifying multivariate time series. Batal et al. (2012) presented a study in which time interval patterns classify multivariate time series.

Finally, as mentioned in the introduction, we have developed a classification framework, the KarmaLegoSification (KLS) framework, which embodies a comprehensive temporal abstraction and classification process, and in particular, the use of TIRPs as classification or prediction features. The KLS framework exploits the KarmaLego algorithm (Moskovitch and Shahar, In Press), which we briefly present in Section 3, for the discovery of frequent TIRPs within a set of entities, each represented as a series of symbolic intervals.

3. Fast Time-Intervals Mining

The discovery of *Time Interval Related Patterns* (*TIRPs*) is computationally highly demanding, since 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 figure 4, with all possible temporal relations among them, requires in theory generating up to $7^{((5^2 - 5)/2)} = 7^{10} = 282{,}475{,}249$ candidate TIRPs. In general, given a TIRP having k time intervals, we will have up to $7^{((k^2 - k)/2)}$ candidate TIRPs.

To overcome this difficulty, we have previously developed *KarmaLego*, a fast algorithm which generates all of the patterns efficiently by extending TIRPs directly, and exploiting the transitivity property of the temporal relations to remove unrealistic candidates (Moskovitch and Shahar, In Press).

To increase the robustness of the discovered temporal knowledge, KarmaLego uses a flexible version of Allen's seven temporal relations. This is achieved by adding an epsilon value to all seven relations, as explained in the next subsection. Furthermore, we also limit the *before* temporal relation by a maximal allowed gap, as proposed by Winarko and Roddick (2007).

To define formally the problem of mining time intervals and relevant measures for the classification task, we first present several basic definitions. These definitions will be used in the description of the methods.

3.1 Definitions

To better understand the SingleKarmaLego methodology, we introduce several key definitions, several of which extend our initial definitions for the KarmaLego framework (Moskovitch and Shahar, In Press).

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* $t_2 - t_1 > \varepsilon$.

Based on the two relations $=^{\varepsilon}$ and $<^{\varepsilon}$ and the epsilon value, a flexible version for all of Allen's seven relations is defined, extending Papapetrou's definition, as shown in Figure 2. 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 temporal relations among two time intervals.

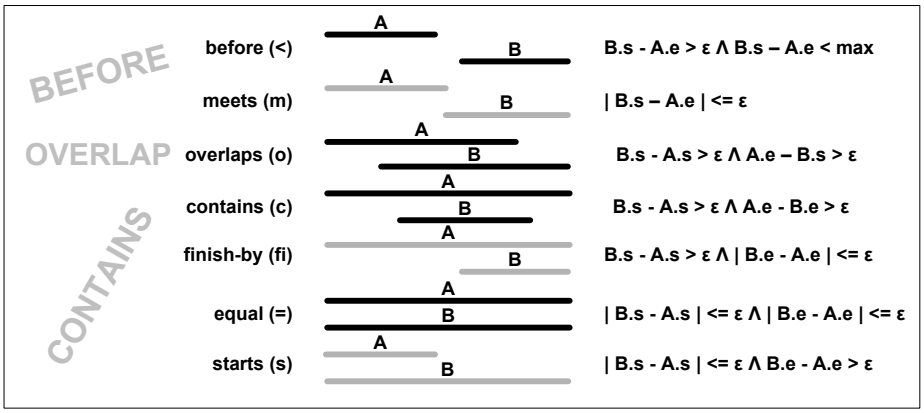

Figure 2. A flexible extension, using the same epsilon value, for all Allen's seven relations, using the same Epsilon for all the relations.

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 figure 2 keeps the set of the temporal relations mutually exclusive. This is indeed true, since the epsilon-extended temporal-relation definitions appearing in figure 2 imply that for any two time intervals, exactly one (epsilon-extended) temporal relation applies.

In addition to a flexible (epsilon-based) semantics for Allen's seven temporal relations, we introduce a set of three abstract temporal relations (shown in figure 4 in grey labels): BEFORE = {before | meet} that is the disjunction of Allen's before and meets, OVERLAP that is Allen's overlap and CONTAIN = {finish-by | contain | start-by | equal} that is the disjunction of finish-by, contain, start-by and equal.

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*, $IS = \{I^1, I^2, .. , I^n\}$, 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 *lexicographic 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, $IS = \{I^1, I^2, .. , I^n\}$, such that:

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

Since in our problem definition the time intervals are ordered lexicographically, we use only the seven temporal relations shown in figure 4.

Definition 5. A non-ambiguous lexicographic *Time Intervals Relations Pattern* (*TIRP*) P is defined as $P = \{I, R\}$, where $I = \{I^1, I^2, .., I^k\}$ is a set of k symbolic time intervals ordered lexicographically and

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

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

Figure 3 presents 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. One potential problem with Definition 5 is that it is purely qualitative; it ignores the precise quantitative durations of the time intervals that are the components of the TIRP. We focus on this problem, and on a possible solution of it, in Section 3.7, when we propose to pre-cluster the time intervals by duration. (In the Discussion, we also consider the other potentially quantitative aspect, i.e. the nature of the temporal relationship itself, such as the duration of the gap in the case of the *before* relation.)

Figure 3 presents the output of a time interval mining process, i.e., an example of a TIRP, represented, for efficiency purposes, as a half matrix. Thus, the half matrix on the right part of figure 3 presents all of the pair-wise temporal relations among the TIRP's symbolic time intervals, ordered lexicographically, that defining it in a canonical, non-ambiguous fashion. Note that *half-matrix* representation (as opposed to a full matrix) is possible due to the fact that each of Allen's temporal relations has an inverse; and that the *canonical* aspect is due to the lexicographic ordering, which leads to a unique half matrix for each TIRP.

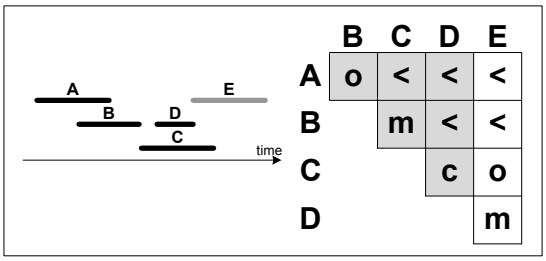

Figure 3. An example of a Time-Interval Related Pattern (TIRP), represented by a sequence of five lexicographically ordered symbolic time intervals and all of their pair-wise temporal relations. On the left, the actual five-symbols TIRP is displayed graphically, while on the right, a half matrix representation is given presenting the pairwise temporal relations between each two symbolic time intervals. Interval E is a candidate symbol that is being added to the current TIRP, and its relations with the other four symbolic intervals are shown in the last column of the half matrix.

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 within which P holds at least once, divided by the total number of entities (e.g., patients) $|E|$: $ver\_sup(P) = |E^P| / |E|$. The vertical support is the term usually referred to as *support* in the context of association rules, itemsets, and sequential mining.

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

Note that in pattern mining, such as association rules, sequential mining and time intervals mining, *support* typically refers to the percentage of entities in the database supporting a pattern, which is actually the vertical support presented in Definition 6.

Since a temporal pattern can be discovered multiple times within a single entity (e.g., the same TIRP appears several times (multiple instances) in the longitudinal record of the same patient), we distinguish between two types of support: the *vertical* support and the *horizontal* support, which represents the number of patterns discovered within the longitudinal record of a specific entity, as defined in definition 7.

Definition 7. The *horizontal support* of a TIRP P for an entity e_i (e.g., a single patient's record), $hor\_sup(P, e_i)$ is the number of instances of the TIRP P found in e_i. For example, the number of times that a particular temporal pattern was found in a particular patient's record.

Definition 8. The *mean horizontal support* of a TIRP P is the average horizontal support of all the entities E^P supporting P (i.e., for all entities that have a horizontal support ≥ 1 for TIRP P).

$$MeanHorizontalSupport(P, E^P) = \frac{\sum_{i=1}^{|E^P|} hor\_sup(P, e_i)}{|E^P|}$$

Definition 9. The mean duration of the n supporting instances of the same *k-sized* TIRP P within an entity e (e.g., within a single patient's record; note that, per Definitions 7 and 8, an entity may have several supporting instances of a TIRP) is defined by:

$$MeanDuration(P, e) = \frac{\sum_{i=1}^{n} \left(Max_{j=1}^{k} I_e^{i,j} - I_s^{i,1} \right)}{n}$$

where $I^{i,1}{}_s$ is the *start-time* (s) of the first time interval in the i instance (among n instances), and the Max operator selects the time interval having the latest *end-time* (e) among the k time intervals of an instance i. Note that according to the lexicographical order (Def 4) the first time interval must have the earliest start-time, while the latest end-time can be of any of the time intervals in the instance.

The horizontal support and the mean duration of TIRP are potentially useful metrics when using TIRPs as features, for classification purposes [Moskovitch and Shahar, 2013].

Definition 10. The time-intervals mining task:

Given a set of entities *E*, described by a symbolic time-interval series *IS*, and a minimum vertical support *min_ver_sup*, the goal of the tasks is to find all the TIRPs whose vertical support is above the *min vertical support* threshold.

3.2 The KarmaLego Algorithm

KarmaLego is a TIRP-discovery algorithm that we have defined and evaluated previously, as part of our research into effective TIRP-based classification (Moskovitch and Shahar, In Press). Here, we summarize the key features of KarmaLego that are relevant to the current study. A full detailed description of KarmaLego can be found in (Moskovitch and Shahar, In Press). The main body of the KarmaLego algorithm consists of two main phases, Karma and Lego (algorithm 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 ∈ T^2 // T^2 is T at the 2nd level
3. Lego(T, t, min_ver_sup) //extend the current TIRP recursively
4. End Foreach
5. return T
6. End

In the first phase, referred to as *Karma* (Algorithm 2), 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 by r, a temporal relation, are discovered, and indexed. Thus, the result of the Karma phase is the frequent symbols appearing in the first level of the enumeration tree – T^1, and the second level of the enumeration tree - T^2 containing all the frequent two-sized TIRPs, which are later extended and used by Lego for the extension process for the discovery of longer TIRPs.

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
5. Index(T^2, < $e.I_{.sym}{}^i$, r, $e.I_{.sym}{}^j$>)
6. End Foreach
7. End Foreach
8. return *T*
9. End

It is important to note here that due to the Karma phase, each query regarding the existence of a particular temporal relation between two specific symbolic-interval instances can be performed in O(1), since all symbolic interval pairs and their respective temporal relations are indexed in a three-dimensional hash table. 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 relevant temporal relation. As we explain later, this structure is also highly useful in SingleKarmaLego.

In the second phase, referred to as *Lego* (Algorithm 3), a recursive process extends the frequent 2-sized TIRPs, referred to as T^2, through an efficient candidate generation method (which exploits temporal transitivity) into a tree of longer frequent TIRPs. These are consisting of conjunctions of the 2-sized TIRPs that were discovered in the Karma phase. Algorithm 4 receives a TIRP *t* that is extended by any of the frequent symbols in T^1, and any temporal relations *r* of the R temporal relations used for TIRP discovery, which is set between the new symbol and the last symbolic time interval. Based on the symbol and the relation *r* a set of candidate extended TIRPs are generated efficiently, exploiting the transitivity of the temporal relations. Then for each candidate a search is performed to retrieve supporting instances that eventually results with the discovery of the enumeration tree of all the frequent TIRPs (figure 4).

Algorithm 3 – Lego(T, t, min_ver_sup)

Input:
T – the enumeration tree after Karma was ran,
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.         C ← Generate_Candidate_TIRPs( t, sym, r)
4.         Foreach c ∈ C // candidates
5.             Search supporting instances(c, T^2)
6.             if( ver_sup(c) > min_ver_sup )
7.                 T ← T U c // c is frequent
8.                 Lego(T, c, min_ver_sup)
9.         End Foreach
10.    End Foreach
11. End Foreach
12. End
```

Note that, for robustness purposes, we are using the flexible version of Allen's temporal relations (see Definition 1). However, *the KarmaLego algorithm is oblivious to the precise definition of temporal relations*. Additionally the KarmaLego algorithm discovers *all* of the frequent TIRPs of an entity (e.g., a patient record), including *all* of the instances of that TIRP over time within that entity. This *completeness* aspect actually enables us later to calculate the novel TIRP representations of horizontal support and mean duration.

Furthermore, in order to make the output of a time intervals mining algorithm such as KarmaLego complete, we must discover *all* of the *horizontally* supporting instances of a TIRP (see Definition 8). This is always the case, because we do not know ahead of time, which of the TIRP instances of size *k*-1 within a specific entity will be followed by a symbolic time interval that will (vertically) support an extended *k*-sized TIRP, and in particular, cannot assume that it will necessarily be the first instance that we discover.

For example, suppose we detect three instances of the 2-sized TIRP <A before B> within a given entity; we cannot know which one of them (if at all) will support the extended 3-sized TIRP, <(A before B) ∧ (A before C) ∧ (B overlaps C)>. For example, if only the third instance of the 2-sized TIRP is followed by a symbolic time interval C (having also the proper relations to A and B), discovering the first instance and stopping will not enable us to discover any evidence for the existence of the 3-sized TIRP within this entity, although it certainly does exist, and will decrease its overall vertical support within the overall data set.

Therefore, it is also important to note that thus, *there is no additional real cost in discovering all of the instances of a given TIRP* within each entity, since finding all of the instances of a given pattern for each entity is actually *obligatory*, in order to ensure *completeness* in the discovery of all [possibly longer] TIRPs, even just for the purpose of determining whether these TIRPs exist!

Note also that determination of such an existence is crucial for classification purposes. An example is when determining whether a given [new] entity to be classified includes a feature, namely, a TIRP, which was previously discovered in the training set. We refer to such an existence as a *binary* representation of that feature. Furthermore, detecting *all* of the TIRPs within the entity is important when the method of representing TIRPs as features uses the Horizontal Support of the entity for that TIRP [see Definition 7] or the TIRP's Mean Duration within that entity [see Definition 9].)

Thus, the KarmaLego algorithm, as well as its single-entity version, the SingleKarmaLego algorithm, are designed to discover (or detect, in the case of SingleKarmaLego) all instance of the TIRP [to be discovered, or that is given, respectively] within each entity.

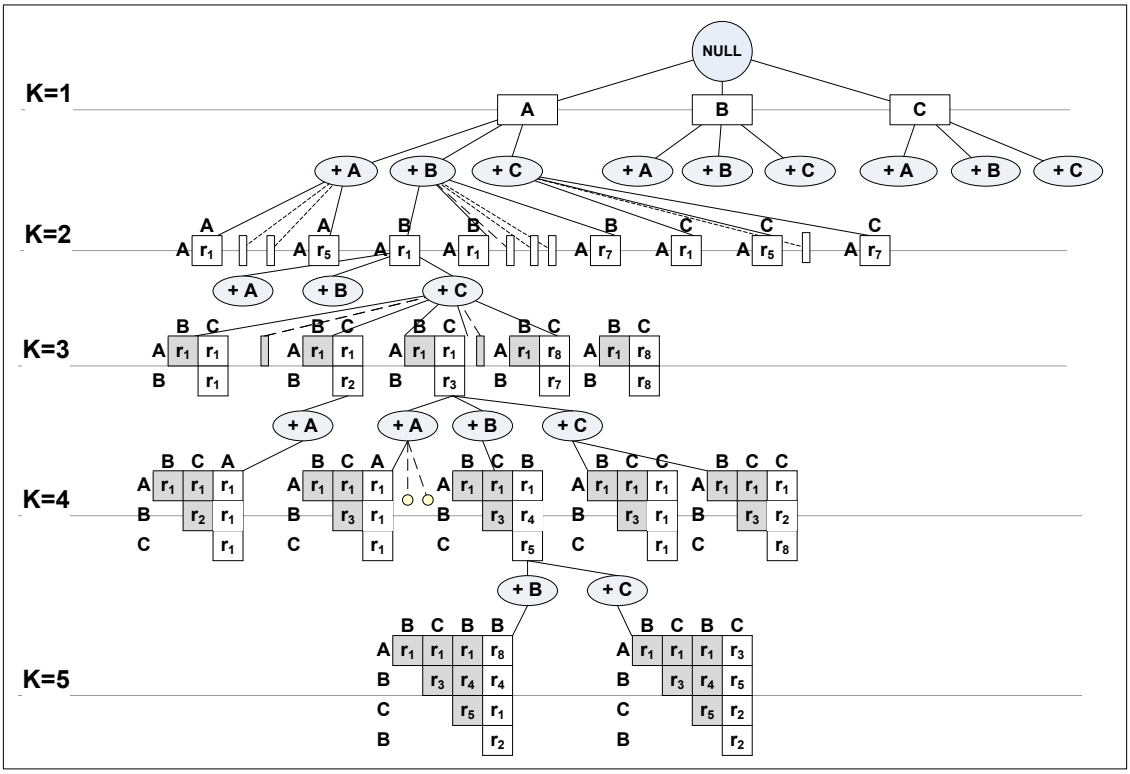

Figure 4. A KarmaLego enumeration tree, in which the direct expansion of TIRPs is performed. Each node represents a TIRP as a half matrix of its relevant temporal relations.

3.3 An Upper Bound for the Complexity of TIRP Enumeration

A rigorous empirical runtime evaluation was previously performed to compare the KarmaLego algorithm's performance to several state of the art algorithms, as demonstrated in our previous study (Moskovitch and Shahar, In Press), highlighting its significant advantages.

As is often the case in mining complex input data algorithms, it is not easy to accurately analyze the KarmaLego algorithm's theoretical runtime complexity, nor is it easy to assess theoretically the effect of the multiple improvements that we added to the algorithm, compared to previous methods. Nevertheless, by making several reasonable assumptions, we can model the worst-case complexity for a general TIRP enumeration algorithm that first indexes the pairwise time intervals, like in Karma, and that enumerates efficiently the TIRPs directly, as described in (Moskovitch and Shahar, In Press). Note, this analysis does not incorporate the reduction in the TIRPs candidate generation performed in KarmaLego.

Assume:
S is the number of the symbolic concept types $S_1..S_s$;
E are the entities $E_1..E_m$, where each entity E_i has N_i ($i = 1..|E|$) symbolic time intervals $I_1..I_{Ni}$;
N = total number of symbolic time intervals for all entities = $\sum N_i$.
R is the number of temporal relations, R≤13; after using a lexicographical ordering, R≤7 (without F, O_i, D, A, M_i, S_i).
L = maximal number of symbolic time intervals in a TIRP, or its maximal length; Note that $L \leq \forall N_i$, ($i = 1..|E|$).

3.3.1 The Setup cost:

First, for the Karma step, we need $O(N_i^2)$ time to determine the temporal relationships among all symbolic time intervals pairs of entity E_i. The total cost over the entire dataset is $\sum N_i^2$

Note that

$$\sum N_i^2 \leq \sum N_i * \text{Max}(N_i) < \sum N_i * N = N \sum N_i = N^2, \quad \text{Since } \forall i, N_i \leq N = \sum N_i.$$

So the cost is bounded by $O(N^2)$, although in practice it would usually be much smaller.

3.3.2 The Enumeration process:

A sub-tree of efficiently enumerated TIRPs is built for each symbol in the root, i.e., all TIRPs whose canonical representation starts with that symbol.

Starting from the root symbol, we branch into up to S symbols, then from each pair of symbols, into up to R temporal relations among the pair of symbols, which resulted with the 2-sized TIRPs.

At every step (i.e., different level in the TIRP enumeration tree) j, after j-1 symbols have been considered for the enumerated TIRP, and the precise temporal relations have been determined among all of the first (j-1) symbols in the TIRP that was found frequent, we need to check in the worst up to R^{j-1} possible temporal relation combinations, due to the potential temporal relations among the new symbol and each of existing symbolic time intervals $I_1..I_{j-1}$ and I_j.

3.3.3 A Complexity Analysis of the Worst Case scenario:

Assume we go in each TIRP tree down to some maximal level L, which depends of course on the minimal vertical support required and on the data set's characteristics.

Since we branch into S symbols each time and check up to R^{j-1} possible temporal relation combinations among the first j-1 concepts and the j^{th} concept at each level on the way, for each tree we need to compute up to $S*R*S*R^2 \ldots *S*R^{L-1}$ relations.

This process needs to be repeated for up to S TIRP trees, whose roots are $S_1..S_s$.

Thus, the total number of temporal constraint checks is:

$$S*(S*R*S*R^2 \ldots *S*R^{L-1}) = S*S^{L-1}*R^{(1+2+\ldots L-1)} = \mathbf{S^L * R^{L(L-1)/2}}$$

But, even with a highly efficient O(1) check for every time intervals pair, checking each temporal constraint for up to N_i symbolic time intervals requires $O(N_i^2)$ time; and this check needs to be performed for all M TIRPs. Assume that in the worst case, we require a vertical support of only 1.

Total temporal constraints checks = $\sum N_i^2 < N \sum N_i = N^2$.

(as we explain below, this worst-case situation is in practice unlikely in the case of the KarmaLego algorithm, unlike in the case of previous algorithms, due to the use of the temporal transitivity property for candidate generation in KarmaLego).

Thus, the overall worst-case complexity of creating the fullest TIRP tree is in theory bounded *in the worst case* by the following expression:

$$O(S^L R^{L(L-1)/2} N^2).$$

(Note that the short Karma-phase $O(N^2)$ setup cost is already overshadowed at this point, and thus can be disregarded.)

Note that the major factor affecting the worst-case complexity is the maximal possible-TIRP length *L*, measured in number of symbolic concepts. Of course, *L* cannot be assessed ahead of time, as it depends on the data set and on the minimal vertical support required. However**,** the analysis does suggest that the most effective way of limiting the potential exponential explosion of a TIRP-detection process is by limiting the maximal TIRP- length *L*, regardless of the data set's properties or the minimal vertical support value.

However, we must emphasize that this worst-case, upper-bound complexity analysis of this theoretical TIRP-enumeration algorithm far overshoots the practical situation, and in particular, far overshoots our own specific implementation of the TIRP-discovery process, namely the KarmaLego algorithm.

In particular, in a recent empirical evaluation of the KarmaLego algorithm, comparing it to several state of the art TIRP-enumeration algorithms, we had demonstrated the highly significant efficacy of the various computational steps that we had taken to reduce the average number of candidates generated for directly extending a TIRP. The main reduction is achieved through the use of the transitivity of temporal relations, such that the number of candidates actually generated is smaller by an order of magnitude, and even two or three orders of magnitude, compared to the number generated by previous algorithms, which do not exploit the transitivity property. In addition, we are using an especially efficient data structure to represent temporal relations between all symbolic interval pairs (Moskovitch and Shahar, In Press).

Thus, the worst case presented here, purely for the reader's interest, is a rather theoretical upper bound for all TIRP-discovery methods, and in practice, as shown by the KarmaLego algorithm's empiricial evaluation, can be significantly much smaller.

4. Methods

In the methods section we present the SingleKarmaLego algorithm that is the focus of this paper, as well as the linear TIRPs detection algorithm commonly used that we will compare SingleKarmaLego runtime to. As part of the methods we will present a rigorous analysis of the complexity of the SingleKarmaLego algorithm and the Sequential TIRPs Detection that will be backed by empirical evidence.

4.1 Detection of Time-Interval Relation Patterns within a Single Entity

As explained in the introductory Section 1, several studies have shown the value of abstracting longitudinal records of time-stamped raw data into a set of symbolic intervals, discovering a set of frequent TIRPs within a training set, and inducing a classifier that uses these TIRPs as features to classify entities within the testing set.

We shall now examine more closely how these TIRPs are detected within each entity. Note that TIRP detection in a single entity is also useful for multiple other purposes.

4.2 The SingleKarmaLego Algorithm

To classify an entity, its symbolic time intervals series must be searched for the features, i.e., frequent TIRPs. When mining a single entity, i.e., finding all of the [frequent] TIRPs for a particular subject (e.g., a patient), the notion of minimal vertical support is meaningless, and thus, all of the relevant TIRPs existing within the record of that entity should be discovered. Moreover, in general, not *all* of the discovered TIRPs are relevant to the classification task: in reality, only the TIRPs that appear in the training data, and were thus defined as features, should be searched, as shown in Algorithm 4. Thus, we modified KarmaLego to be applied to a single entity, a process that we refer to as SingleKarmaLego.

Algorithm 4 – SingleKarmaLego

Input:

entity_sti – the single entity's symbolic time intervals records; that is, e.I

T – the enumeration tree of the [full] training set

Output: *Single_T* – an enumerated tree of TIRPs

1. *Single_T* ← SingleKarma(*entity_sti*, *T*, *epsilon*)
2. Foreach $t \in Single\_T^2$ // $Single\_T^2$ is *Single_T* at the 2nd level
3. SingleLego(Single_*T*, *T*, *t*, *2*, *epsilon*)
4. End Foreach
5. return *Single_T*
6. End

Thus, in the classification phase, each single entity *e* has to be mined separately to search for occurrences of the specific TIRPs in *P*, as will be shown in Algorithm 4 – SingleKarmaLego. Note that since the vertical support is meaningless, there is no stopping criterion for the algorithm and any discovered TIRP will be "frequent". However, since we are interested in discovering only the TIRPs that were discovered in the training phase, the algorithm should search for these TIRPs only. In Algorithm 4, the input is a set of TIRPs *P*, which was discovered in the training set (the unification of all the classes), which is actually the enumeration tree that was discovered. Thus, the goal is to follow the enumeration tree which is represented by P in order to generate and discover all the existing TIRPs in the entity. The mining process is actually the same as was described in KarmaLego, but without considering the minimal vertical support.

Algorithm 5 is similar to Algorithm 2 (Karma), but instead of generating all the possible TIRPs, it searches only for the TIRPs at the first and the second level in the given enumeration tree T that was discovered in the training set phase. After that, for all the TIRPs at the third level that appear in T, the SingleLego algorithm is called to further generate and search the extended TIRPs of this path in the tree T.

Algorithm 5 – SingleKarma

Input:

entity_stis – the symbolic time intervals of the single entity; that is, e.I

T – a [full] set of TIRPs that were discovered in the training phase;

Epsilon – the size of epsilon in the temporal relations;

Output: *Single_T* – an enumerated tree of TIRPs

1. *Single_T* ← Ø
2. Foreach $I^i, I^j \in$ entity_stis $\wedge$ $i<j$
3. $r \leftarrow$ the temporal relation among I^i, I^j given *epsilon*
4. if $< I_{.sym}^{\ i}, r, I_{.sym}^{\ j} > \in T^2$
5. Index($Single\_T^2, < I_{.\mathrm{sym}}^{\ i}, r, I_{.sym}^{\ j} >$) //*indexed as in Algorithm 2*
6. End Foreach
7. return *Single_T*; //*return a tree of up to two levels*
8. End

Algorithm 6 is similar to algorithm 3 (Lego), but instead of generating all of the frequent TIRPs, it generates the TIRPs that appear in T, which were discovered in the training phase and are required for the classification, and searches for them in the second level (Single_T2) that was created in SingleKarma.

The extended TIRP *t* is an object containing the TIRP properties, such as: a vector of symbols, a vector representing the half-matrix of the temporal relations, and a list of supporting instances (which is basically a vector of pointers to the sequence of time intervals), as described in more detail elsewhere (Moskovitch and Shahar, In Press). The TIRP's object *t* data structure supports a highly efficient O(1) time query regarding any particular relationship between two symbols in Single_T^2, without starting the detection of the TIRP within the entity's set of symbolic interval from scratch, as a naïve TIRP detection method might do.

If the supporting instances of a TIRP are found, SingleLego searches for its extended TIRPs, as long as they appear in T, recursively. Thus, in each extension, new TIRPs are created using an extended vector of symbols (i.e., adding the next, k^{th} symbol), a vector of temporal relations (i.e., the temporal relations between the candidate. Note that the TIRP's horizontal support within the entity can increase, depending on the particular set of symbols prevalent for each entity.

Algorithm 6 – SingleLego

Input:

Single_T – the enumerated tree of the entity (sent by reference)
T – the enumeration tree of the training set,
t – the extended TIRP;
level – the level of the enumeration in the tree

Output: void

1. Foreach $t^{level+1} \in T$
   (Note that like t, $t^{level+1}$ is an object that stores, for any two symbols, all of the symbolic interval instances satisfying all of the relations between them)
2. $t^{level+1}.insts \leftarrow$Search supporting instances of $t^{level+1}$ in *Single_T*
3. if $|t^{level+1}.insts| > 0$ *//found at least one instance of this level in T, within Single_T*
4. SingleLego(*Single_T, T*, $t^{level+1}$, *level+1*) *//extend by another level*
5. End Foreach
6. End

The single mining process results with an enumeration tree Single_T, which is a sub-tree of T, containing all of the TIRPs that were discovered in the given entity *e*. Then, as we show in Section 4, a vector of features (TIRPs), having one of the TIRP representations, is created. The vector of features is then used for classification. Note that each TIRP of size $k \geq 2$ is considered a feature; i.e, not just the longest TIRPs, or the leaves of the TIRP tree. Once all of the size $k \geq 2$ TIRPs are detected for a single entity, their horizontal support and mean duration are calculated, to represent them for classification purposes.

5 A Complexity Analysis of the SingleKarmaLego algorithm, versus that of a Sequential TIRPs Detection algorithm

The SingleKarmaLego (SKL) algorithm is significantly superior, computationally, to a simpler, sequential TIRP-detection method that attempts to find each feature (i.e., TIRP) within a vector of symbolic time interval instances of a given entity (sorted lexicographically). The advantage can be demonstrated even when detecting only 2-sized TIRPs, and looms much larger as the entity is queried for longer and longer TIRPs within the SingleLego recursive procedure.

The first and main advantage of the SKL algorithm stems from the one-time, preprocessing indexing step performed for each queried entity in SingleKarma, in which the Single_T^2 tree, representing all of the relations between two symbolic time intervals that appear in the entity, is created, and is represented as a matrix indexed by the two symbols. (Recall that, as in the KarmaLego algorithm, we represent the temporal relations amongst all symbolic time intervals pairs, using a matrix of all the symbols pairs, having a vector of 7 entries for each temporal relation, in which a HashMap indexes all pairs of symbolic time intervals instances. The HashMap Key is the first symbolic time interval and the Value is a list of symbolic time intervals following the Key symbol that corresponds to the given temporal relation between them). The result is a highly efficient structure (actually, a three-dimensional array) for the purpose of query and retrieval of 2-sized TIRP instances, which supports, during the TIRP extension process, a very fast (O(1)) way of determining whether a given relation indeed exists between symbolic time interval instances of the two symbols.

The second advantage of the SKL algorithm stems from the fact that each extension of the length of a query TIRP, beyond two symbols, is performed directly and is also in O(1), given the extended TIRP and its detected instances; since, as in KarmaLego, an index of the 2-sized TIRPs is created by SingleKarma, adding the $l^{th}+1$ symbol to an *l*-sized requires only a single query on the index in O(1).

We will now analyze and compare the overall complexity of both a simple sequential TIRPs-detection algorithm and the SKL algorithm, which includes implicitly the worst-case analysis, and then look at the "practical" behavior of both, which considers also the mean ("average") case, to determine when precisely is the SKL algorithm superior to a sequential one. We will show that the SKL algorithm is superior both in the overall, usual sense of time complexity, as well as in the "average" case.

5.1 The Sequential TIRPs–Detection Algorithm

To make our case clearer, we shall start by defining more formally the sequential TIRPs-detection (STD) algorithm. Algorithm 7 describes a typical STD algorithm. The algorithm accepts a vector of the single-entity symbolic time intervals (entity_stis) and a list of TIRPs to detect (tirpsList). The algorithm detects all of the instances for each of the TIRPs (line 1). For each TIRP, it starts by going over all of the symbolic time intervals (line 2). If the first symbol of the current TIRP (tirp.s[0]) was detected, a While loop starts to look for the TIRP's next symbols, and verifies that their temporal relations with the previous detected symbolic time intervals are the same as in the TIRP temporal relations definition (tirp.r). If the time duration between the i-interval and the j-interval is larger than max_gap, the search for instances is stopped.

Algorithm 7 – Sequential TIRPs Detection

Input:

entity_stis – the symbolic time intervals of the single entity;

tirpsList – list of TIRPs to detect

Epsilon – the size of epsilon in the temporal relations;

max_gap – the maximal time duration allowed in the before temporal relation

Output: *tirps (containing their detected instances)*

```
1. Foreach tirp in tirpsList
2.     Foreach I^i ∈ entitiy_stis
3.         If( I_sym^i == tirp.s[0])
4.             j ← i
5.             tIdx ← 1
6.             instance ← I_sym^i
7.             while(j < |entitiy_stis| && tIdx < tirp.size)
8.                 If(I_sym^j == tirp.s[tIdx])
9.                     If any relations among I_sym^j and instance.Is ≠ tirp.r
10.                        break
11.                    Else
12.                        instance ← instance ∪ I_sym^j
13.                        If(tIdx == tirp.size)
14.                            tirp.instances ← tirp.insts ∪ instance
15.                            tIdx ← 1
16.                            instance ← I_sym^i
17.                    EndIf
18.                If(I_sym^j – I_sym^i > max_gap); i,j, are indices of time intervals
19.                break
20.            Endwhile
21.        EndIf
22.    EndForeach
23. EndForeach
24. End
```

5.2 Comparing the worst case complexity of the SKL algorithm to that of the STD algorithm

Let us denote by *m* the number of TIRPs discovered during the training phase, which need to be queried and detected within the current set of *n* symbolic time intervals of the single entity to be classified; by *l* the number of symbols in a TIRP to be detected within the entity; and by *S* the total number of different symbolic concepts, or symbols, in the dataset.

To compute complexity, we will count the number of operations that are needed to perform in each TIRP-query (i.e., each detection for a given TIRP within the entity's record). An operation is a computation of the temporal relation between two given time intervals, a single lookup into a table, or a hash function call.

In the simplest case, when l=2, the SingleKarma algorithm requires a constant preprocessing one-time set-up cost of $n^2/2$, or more precisely, $n*K/2$ operations to index *all* pairwise relations in the entity n symbolic time intervals; K represents the mean number of time intervals within the max_gap duration, G. (K is different for each data set, as is G, which represents the domain's *max_gap* parameter [which defines the maximal time duration that still allows bridging the gap between two symbolic time intervals to determine their temporal relation]). Thus, query and retrieval for a given symbolic-interval pair and temporal relation (i.e., determining if there is such an instance, and if so, retrieving at least one example) requires only O(1) operations, by looking up the index of Single_T^2. The m queries are therefore performed at m*O(1) time.

Thus, when l=2, the cost for SKL of m TIRP-queries is

O($n*K$)+m*O(1) operations.

That is, an *additive* growth behavior, relative to the number of TIRPs to search for.

Versus that, even for l=2, a sequential TIRP-detection algorithm requires at least O($n*K$) operations to detect, or prove the nonexistence of a single 2-sized TIRP (two related symbolic time intervals) within a vector of time intervals, even though the intervals are sorted lexicographically.

Thus, a sequential algorithm requires even for l=2 symbols per TIRP, for m TIRP-queries,

O($n*K*m$) operations.

That is, a *multiplicative* growth behavior, relative to the number of TIRPs to search for.

However, when l>2, the advantage of SKL looms much larger, both due to the key pre-processing step, which creates the Single_T^2 and reduces the cost of each pair-query, and due to the efficiency of the SingleLego procedure itself. That is because, in general, for an l-sized TIRP, for a naïve algorithm, once we try to extend the TIRP by another symbol, we need to perform for each of the m TIRP-queries, (l*(l-1)/2) operations (one for each pairwise temporal relation), to disambiguate all of the pairwise temporal relations among all components of the given TIRP, since in STD we do not have the SingleKarma indexing phase, as in SKL.

Thus, overall, for any l, to answer m l-sized TIRP-queries, using a sequential algorithm, we need

O($n*K$)*m *(l(l-1)/2) operations.

Note that, apart from its inefficiency in checking each pairwise relation between two symbols (i.e, asking a size-2 TIRP-query), a naïve algorithm also does not exploit the fact that a TIRP-query was previously asked about an (l-1)-sized TIRP; but in SingleLego, both of these advantages prove their benefit.

First, in SKL, verifying that a pair of symbolic time interval instances exists for each pairwise relation between two symbols that is checked while extending a given TIRP, requires only O(1), due to the SingleKarma phase. This is because such an operation looks up, as explained, the three dimensional array of pairwise relations, Single_T^2. (The number of overall operations might be incremented slightly by the need to search, even after given a direct index to a cell of the Single_T^2 matrix, among several instances of the given symbol pair within the single entity).

Second, for each l-sized TIRP, only (l-1) relations need to be checked, comparing the new symbolic time interval at level l to the previous (l-1) symbolic time intervals (as illustrated in the last column of the half-matrix in figure 5). Note that, as a result of the recursive process, when the l^{th} symbol of an l-sized TIRP (feature) is added, its pairwise temporal relations with the rest of the (l-1) symbols are being queried for existence in the entity's record. Thus, the specific corresponding (l-1) sized TIRP (including the specific symbolic time interval instances from which it is composed) is being held in memory, so the only checks needed to be made consist of comparing the new symbolic time interval to the previous (l-1) symbolic time intervals. As explained, each such check exploits the Single_T^2 data structure and requires only O(1) operations.

Thus, overall, for any l, to answer m l-sized TIRP-queries, using **SKL**, we need only

O(n*K) + m*(l-1)* O(1) operations.

Again, we see that using SKL leads, in general this time, to an *additive* growth instead of a *multiplicative* growth relative to the number of TIRPs to find in the entity.

To sum up, sequential detection requires approximately **O(n*K*m*l^2**) operations, while using SKL requires approximately only **O(n*K + m*l**)** operations, a highly significant computational advantage, which looms larger as the number of TIRP-queries, *m,* increases (in theory, that number might increase exponentially, relative to the number of symbols, as a result of the training phase), and as the number of symbols per TIRP, *l,* increases (note that *l* is really limited only by the number of time intervals within the entity of the training set that has the maximal number of intervals).

5.3 Comparing the average-case complexity of SKL to that of Sequential TIRPs Detection

We now continue by providing an analysis of the problem that examines the practical question, *when* using SKL is better, in the average case, over a sequential TIRPs-detection method.
Given an entity having *N* symbolic time intervals ordered lexicographically, and as before, *G* representing the domain's *max_gap* parameter (which limits the before temporal relation, and represented here by a mean number of symbolic time intervals occurring during the max_gap time duration), the task is to detect all the instances, of a given *M* (queries) TIRPs, which hold within the entity's symbolic time intervals. (These TIRPs were presumably discovered in the training set, and are now used as features to classify the new entity).
Since this is a practical "average-case" analysis, we will represent again by the integer *K* the typical mean number of time intervals within the *max gap G* time duration.
For further simplicity's sake, we will first assume that all of the *M* (queries) TIRPs are two-sized TIRPs. We will show that even for this simplified case, SKL is usually faster, and thus certainly for more complex cases (which approach the above analyzed worst case scenario, in which we have already proven SKL's advantage).

5.3.1 Detection using a Sequential TIRP-Detection algorithm

To detect all of the instances of a given set of *M* TIRPs within a given entity, a sequential method will have to run through the entire list of the entity's *N* time intervals to detect, for each potential TIRP, the first time interval; and then check up to an additional *K* time intervals (on average) to detect the second time interval of the same TIRP – each time the first symbol was detected in the *N* time intervals. Thus, discovering all of the instances of a given TIRP will require $N + i*K$ operations, in which *i* is the number of the times in which the first symbol of the pair was detected*;* $i \leq N$. (*i* can be considered as an upper bound on the number of potential instances of the TIRP that can be found, once we find the first symbol; on average, $i = N/S$, given *S* symbolic concepts).
Since the task is to detect *all* of the instances of the *M* given TIRPs, the number operations required will be

(*N* + *i*K*)**M*** operations.

(Note that, in the worst case, in which every interval was a potential first component in the TIRP, we need $(N + N*K)*M$ operations.)

5.3.2 Detection using the SKL TIRP- detection algorithm

To detect all of the instances of a TIRP, SKL first indexes all of the time intervals pairs by scanning the *N* time intervals, which requires $N*K$ operations. From this point and onwards, detecting all of the instances of each of the 2-sized TIRPs requires only O(1), due to our hash-table matrix representation, as explained regarding the time-relations data structure of the KarmaLego and SKL algorithm.

Thus, detection within a given single entity, of all of the instances of M given TIRPs, will require

N**K+M operations.

5.3.3 Determining when SKL is superior to a Sequential TIRPs Detection Algorithm

We can now ask, in addition to determining the overall complexity of both algorithms, precisely *when* using SKL is superior to the use of the STD algorithm? That is, when does the STD

algorithm require more operations?

Our question is reduced, given the previous analysis, to asking:

When does the following inequality hold?

$$(N+i*K)*M > N*K+M$$

i.e.,

$$NM+i*K*M > N*K+M$$

To answer the question, we shall look at two cases.

First, in the typical case, $M > K$.

(Note that during the TIRP-discovery process in the training set, the number of TIRPs to detect, *M,* is related to the number of TIRPs initially discovered, and thus might grow polynomially or even exponentially relative to the number of symbolic interval instances per entity; while the mean number of intervals per maximal gap *K*, is a database-related constant, and can grow only in a sub-linear fashion, relative to *N*.

Thus:
If indeed $M>K$, as is the usual case, then the inequality is immediate, by comparing the first and second components of the left and right hand side, respectively (since M>K and, in the typical case, $i*K>1$).

Second, even if $M < K$ for the inequality to hold, it is sufficient that

$$M/K>(N-iM)/(N-1),$$

as can be seen by rearranging the symbols:

Assume that

$$(N+i*K)*M = NM+i*K*M > N*K+M;$$

Thus,

$$NM-M > NK-iKM;$$
$$M(N-1) > K(N-iM);$$
$$M/K > (N-iM)/(N-1).$$

We will now show that even if $M <K$, this inequality will hold in the average case. Recall that on average, $i = N/S;$ thus, all we need to show in the average case is that

$$M/K > (N-(N/S)M)/(N-1);$$

By approximating N-1 by N, and then dividing the enumerator and denominator of the right hand side by *N,* we get:

$$M/K > 1-(M/S).$$

But this inequality is sure to hold in the average case, since $M/K > 0$, while the number of TIRPs M grows polynomial or exponentially in the number of concepts S, so $M/S>1$.

Furthermore, even if M is simply equal to the number of concepts S, e.g., the only TIRPs are the original single symbols,

$$M/K > 1-(1/1) = 0$$

Still holds.

Thus, not only is SKL clearly preferable in the worst case to a simpler sequential algorithm, but SKL is also usually preferable in the typical, "average" case to a simple Sequential TIRPs Detection algorithm, even in the simplest case of 2-sized TIRPs.

Finally, note that in the usual case of using SKL, extending a 2-sized TIRP to the more typical 3-sized TIRP (and even into a 4-sized, 5-sized… etc TIRP) will require again only O(1) [only a single operation](multiplied by the l operations needed to check on internal interval relations), while using the Sequential TIRPs Detection algorithm will require, scanning from the beginning of the time intervals list, $N*K^2$ operations (multiplied by the $l(l-1)$ operations needed to disambiguate all internal interval relations). Thus, we quickly approach the worst-case scenario analyzed previously.

Moreover, unlike several algorithms appearing in the literature, if SKL notes that a TIRP has no instances, it will not try to extend that TIRP.

6 An Empirical Evaluation: Comparing the SKL Runtime to that of STD

6.1 Experimental Plan

We have implemented the sequential TIRP-detection algorithm described previously, and the SKL algorithm, and tested both on several data sets, for the purpose of detecting a set of given TIRPs within several given entities. We wanted to evaluate the methods on conditions that are typical in classification evaluation experiments, in which a batch of dozens, or more, entities are classified in a k-fold cross validation study, and there are dozens of TIRPs presented to be detected, as was the case in a classification study in several different medical domains that we performed (Moskovitch and Shahar, 2013).

We present sample runtime results for the three widely different datasets: The Diabetes data set, which includes a mean of 51 symbolic time intervals in each entity; the Hepatitis data set, which includes a mean of 174 symbolic time intervals in each entity; and the ICU data set, which includes a mean of 292 symbolic time intervals in each entity. We tested the algorithms when getting as input 40, 50, 60, 80 and 100 TIRPs to detect, and 25, 50, 100, or 200 entities in which to detect these TIRPs, to demonstrate the increasing difference in runtime of the Sequential TIRPs Detection algorithm, which requires longer and longer computation times, in comparison with the SKL algorithm, whose runtime grows only slightly as the number of entities significantly increases. In addition, we wanted to demonstrate the influence of the max_gap value on the runtime. Thus, we used two values, 5 and 10 time units.

Note that 200 entities is actually a rather small number; typical entities might be patients to be classified for existence of a disorder, or mobile devices to be classified for malware detection purposes. Their number will be much larger, of course, so the accumulating relative advantage for detecting TIRPs in each single entity will be in practice very large, and thus highly significant.

6.2 Datasets

We compared the runtime of the two algorithms on three distinctly different data sets from the medical domain introduced bellow.

6.2.1 The Diabetes Dataset

The diabetes dataset, provided as part of collaboration with Clalit Health Services (Israel's largest HMO) contains data on 2004 patients who have type II diabetes. The data were collected each month from 2002 to 2007. The dataset contains six concepts (laboratory values or interventions) recorded over time for each patient: hemoglobin-A1c (HbA1C) values (indicating the patient's mean blood-glucose values over the past several months), blood glucose levels, cholesterol values, and several medications that the patients purchased: oral hypoglycemic agents (diabetic 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. The diabetes dataset was used for the runtime evaluation and for the clustering experiments.

6.2.2 The Intensive Care Unit (ICU) Dataset

The ICU dataset contains multivariate time series of patients who underwent cardiac surgery at the Academic Medical Center in Amsterdam, the Netherlands, between April 2002 and May 2004. Two types of data were measured: static data including details about the patient, such as age, gender, type surgery the patient underwent, whether the patient was mechanically ventilated more than 24 hours during her postoperative ICU stay, and high frequency time series, measured every minute over the first 12 hours of the ICU hospitalization.

The data include: mean arterial blood pressure (ABPm), central venous pressure (CVP), heart rate (HR), body temperature (TMP), and two ventilator variables, namely fraction inspired oxygen (FiO2) and level of positive end-expiratory pressure (PEEP), and low frequency time-stamped data, including base excess (BE), cardiac index (CI), creatinine kinase MB (CKMB) and glucose. For the knowledge-based state abstraction, we used the definitions of Verduijn et al (2007). In addition, the data were abstracted with computationally unsupervised and supervised abstraction methods. (See the experimental results section).

6.2.3 The Hepatitis Dataset

The hepatitis dataset contains the results of laboratory tests performed on patients who hepatitis B or C, who were admitted to Chiba University Hospital in Japan. Hepatitis A, B, and C are viral infections that affect the liver of the patient. Hepatitis B and C chronically inflame the hepatocyte, whereas hepatitis A acutely inflames it. Hepatitis B and C are especially important, because they involve a potential risk of developing liver cirrhosis and/or carcinoma of the liver.

The dataset contained time-series data regarding laboratory tests, which were collected at Chiba University hospital in Japan. The subjects included 771 patients of hepatitis B and C who had tests performed between 1982 and 2001. The data included administrative information, such as the patient's demographic data (age and date of birth), pathological classification of the disease, date of biopsy, result of the biopsy, and duration of interferon therapy. Additionally it included the temporal records of blood tests and urinalysis. These tests can be split into two sub-categories, in-hospital and out-hospital test data. In-hospital test data contained the results of 230 types of tests that were performed using the hospital's equipment. Out-hospital test data contain the results of 753 types of tests, including comments of staff members, performed using special equipment at the other facilities. Consequently, the temporal data contained the results of 983 types of tests. We selected eleven variables which were found most frequent (occurring in most of the patients), including: Glutamic-Oxaloacetic Transminase (GOT), Glutamic-Pyruvic Transminase (GPT), Lactate DeHydrogenase (LDH), TP, ALkaline Phosphatase (ALP), Albumin (ALB), Total BILirubin (T-BIL), Direct BILirubin (D-BIL), Indirect BILirubin (I-BIL) and Uric Acid (UA).

6.3 Results

Figures 5-10 show the runtime comparison of the SingleKarmaLego (SKL) algorithm versus the Sequential TIRPs Detection (STD) algorithm on the three datasets, with two sizes of max_gap of 5 and 10 time units in the domain (months for Diabetes; days for the Hepatitis; and seconds for the ICU). On each max_gap the algorithms were ran on 50, 100, and 200 entities, with 40, 60, 80, 100, and 120 TIRPs to detect. Each graph shows the runtime in second on the vertical axis and the horizontal axis shows both the number of TIRPs at the top and the number of entities in the bottom.

In all of the graphs, it can be clearly seen that while SKL's runtime is growing slightly with the increase in the number of entities and the number of TIRPs, the runtime of STD is increasing quickly, especially when applying it to 200 entities.

The slow-down in the STD computation time is even more apparent when the maximal gap is increased from 5 to 10 time units.

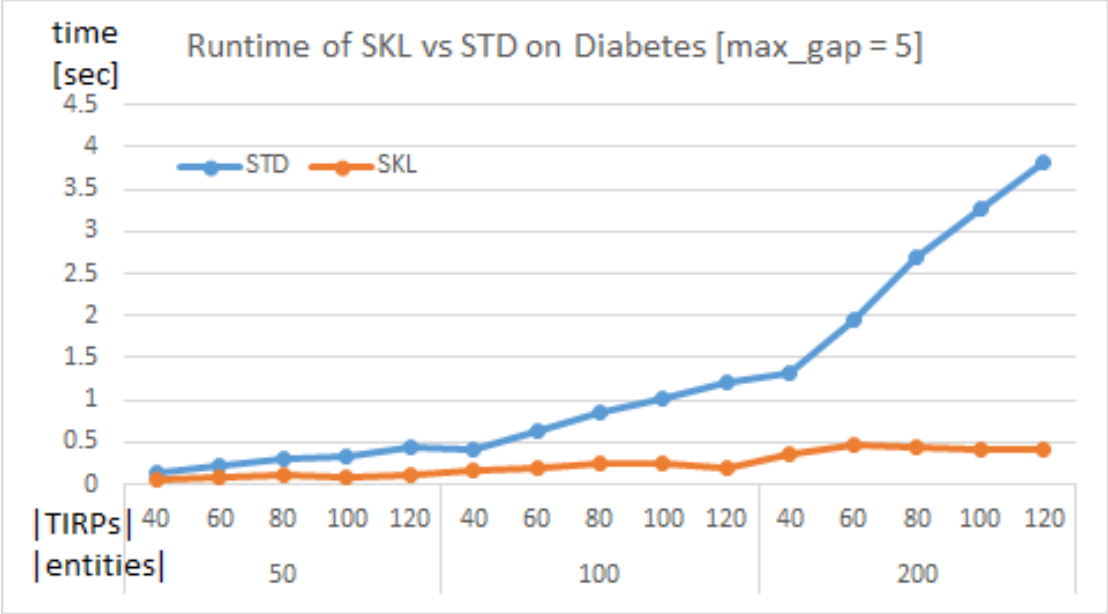

Figure 5. A runtime comparison of the SingleKarmaLego (SKL) and Sequential TIRP Detection (STD) algorithms to detect a set of TIRPs within multiple given entities and a number of TIRPs to detect, on the Diabetes dataset. The maximal gap here was 5 domain time units (here, months). While SKL's runtime is growing slightly with the increase in the number of entities and the number of TIRPs, the runtime of STD is increasing quickly, especially when applying it to 200 entities.

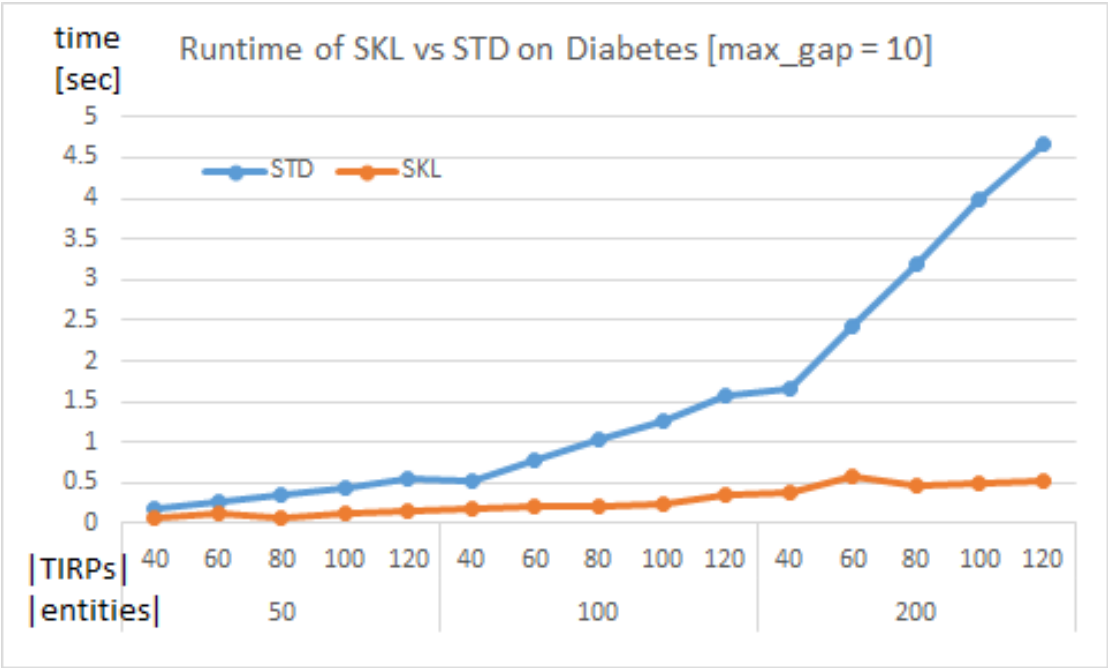

Figure 6. A runtime comparison of the SingleKarmaLego (SKL) and Sequential TIRP Detection (STD) algorithms to detect a set of TIRPs within multiple given entities and a number of TIRPs to detect, on the Diabetes dataset. The maximal gap here was 10 domain time units (here, months). While SKL's runtime is growing slightly with the increase in the number of entities and number of TIRPs, the runtime of STD is increasing quickly already when applying it to 100 entities. Compared with using a max_gap of 5 time units, the STD algorithm slows down sooner.

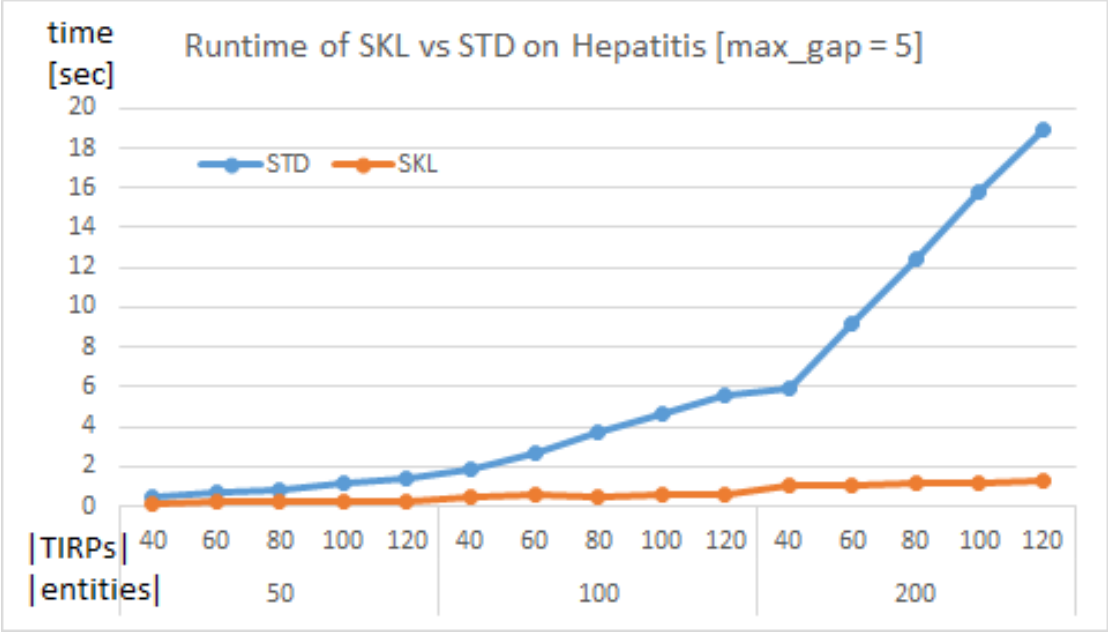

Figure 7. A runtime comparison of the SingleKarmaLego (SKL) and Sequential TIRP Detection (STD) algorithms to detect a set of TIRPs within multiple given entities and a number of TIRPs to detect, on the Hepatitis dataset. The maximal gap here was 5 domain time units (here, days). While SKL's runtime is growing slightly with the increase in the number of entities and number of TIRPs, the runtime of STD is increasing quickly, especially when applying it to 200 entities.

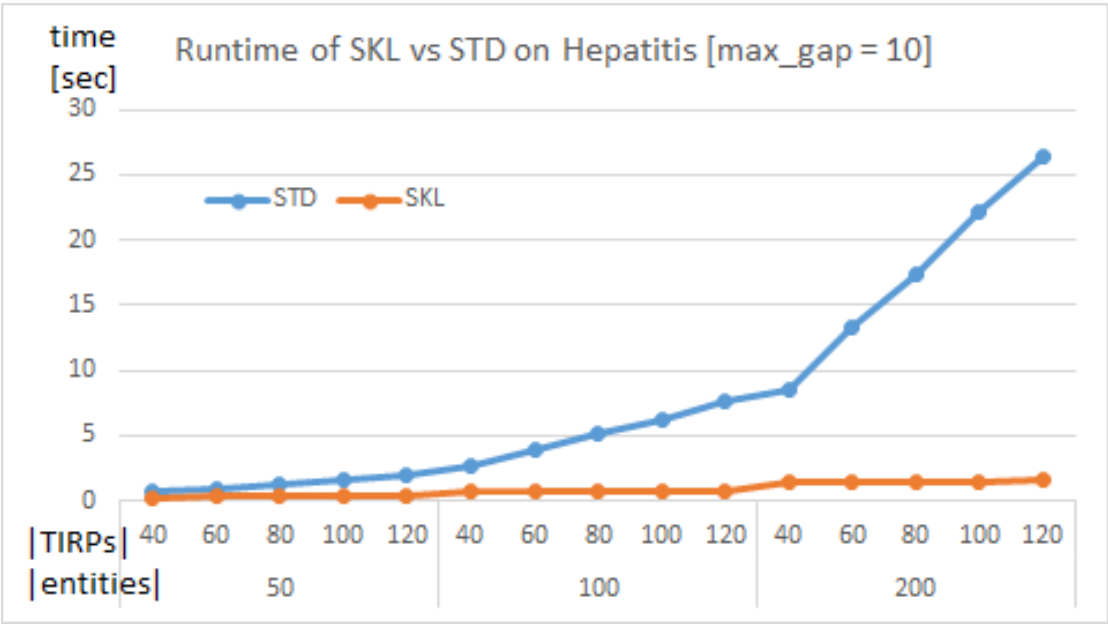

Figure 8. The runtime comparison of the SingleKarmaLego (SKL) and Sequential TIRP Detection (STD) algorithms to detect a set of TIRPs within multiple given entities and a number of TIRPs to detect, on the Hepatitis dataset. The maximal gap here was 10 domain time units (here, days). While SKL's runtime is growing slightly with the increase in the number of entities and number of TIRPs, the runtime of STD is increasing quickly already when applied to 100 entities. Compared with using a max_gap of 5 time units, the STD algorithm is slowing down sooner (note the different time scale).

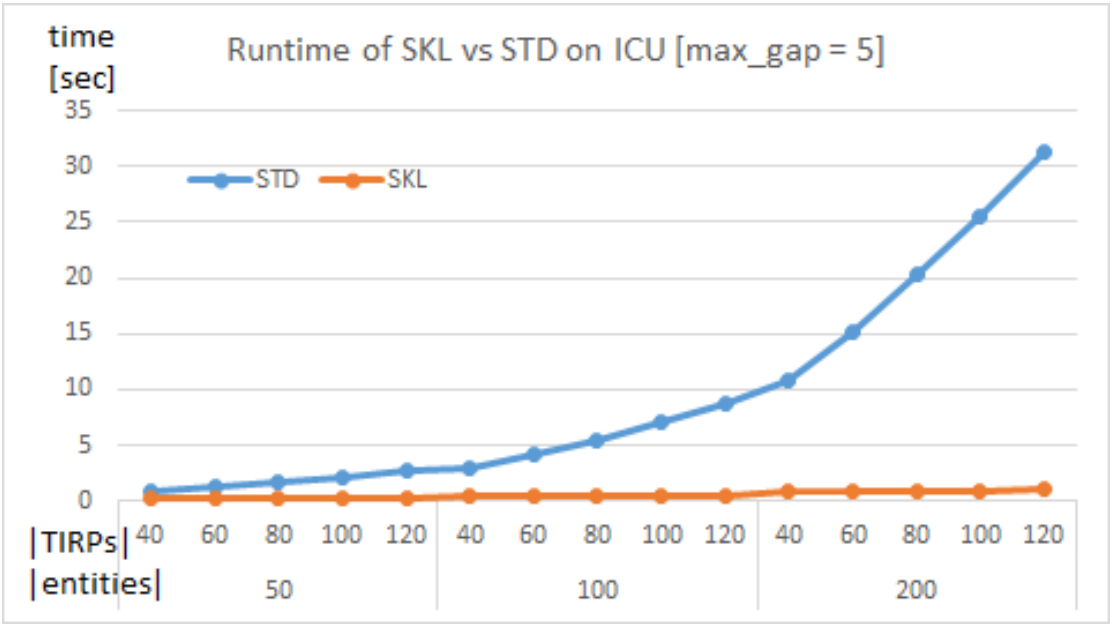

Figure 9. The runtime comparison of the SingleKarmaLego (SKL) and Sequential TIRP Detection (STD) algorithms to detect a set of TIRPs within multiple given entities and a number of TIRPs to detect, on the ICU dataset. The maximal gap here was 5 domain time units (here, seconds). While SKL's runtime is growing slightly with the increase in the number of entities and number of TIRPs, the runtime of STD is increasing quickly, especially when applied to 200 entities.

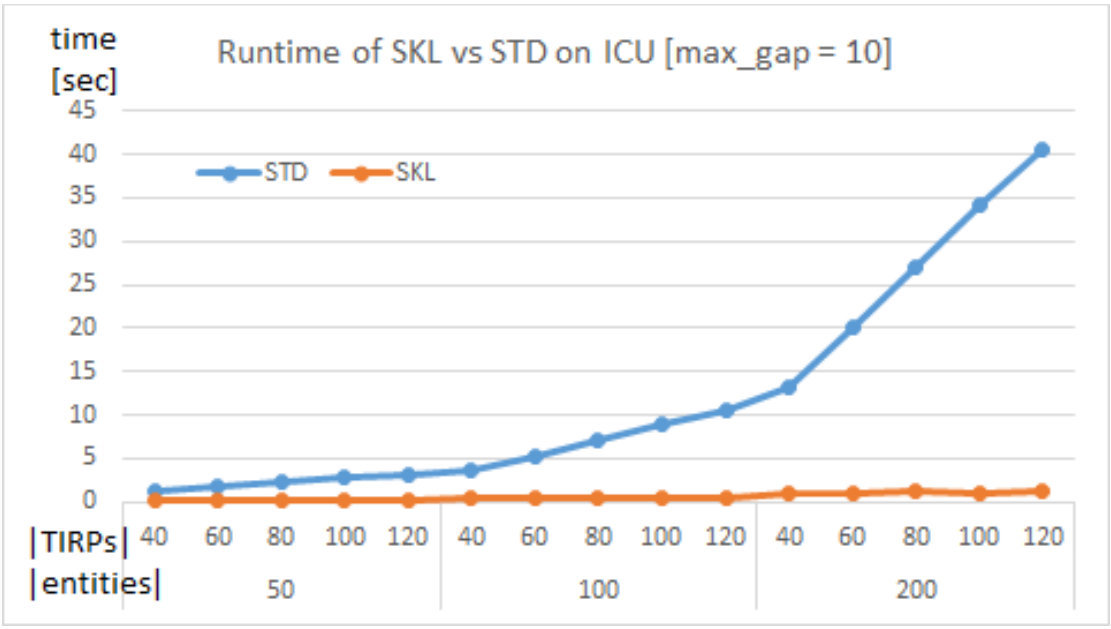

Figure 10. The runtime comparison of the SingleKarmaLego (SKL) and Sequential TIRP Detection (STD) algorithms to detect a set of TIRPs within multiple given entities and a number of TIRPs to detect, on the ICU dataset. The maximal gap here was 10 domain time units (here, seconds). While SKL's runtime is growing slightly with the increase in the number of entities and number of TIRPs, the runtime of STD is increasing quickly already when applied to 100 entities. Compared to the max_gap = 5, the STD algorithm is slowing down sooner.

6. Summary and Discussion

We introduced a new fast TIRP detection algorithm, the SingleKarmaLego (SKL) algorithm, whose objective is to detect multiple TIRPs within one or more single entities represented as a series of symbolic time intervals. One of the motivations for fast detection of TIRPs is the need to detect multiple TIRPs as features, as part of the task of classification of multivariate temporal data, the value of which has been demonstrated in multiple studies, including our own. Other reasons for fast detection of TIRPs exist whenever a set of complex temporal patterns must be searched within a given interval-based entity.

We then analyzed the complexity of the SKL algorithm, comparing both its worst-case behavior and its average-case behavior to those of a rather intuitive sequential TIRPsdetection algorithm, and showing the significant theoretical superiority of the SKL algorithm in both cases, in spite of its relatively high initial setup cost (due to the SingleKarma indexing phase).

We then performed a series of experiments, comparing the runtime behavior of the SKL algorithm to that of the sequential TIRPs-detection algorithm, on three very different sets, with highly varying typical temporal granularities and sampling rate, of longitudinal, multivariate clinical data. The results have demonstrated empirically the significant speed-up in run time provided by the SKL algorithm, compared to that of the sequential-search algorithm's runtime. The speed-up was demonstrated for two settings of the max_gap parameter, which implied an increase in the runtime for both algorithms for the larger maximal gap. The number of entities in which to detect the TIRPs varied gradually, from 50 to 200. Running the methods on a larger number of entities serves two purposes: to demonstrate the runtime for a large number of entities, as happens in practice when performing a classification evaluation run, and to obtain a better estimate of the mean runtime difference per entity, by implicitly using various types of entities. Increasing the number of the entities increased the runtime for both methods, but in the case of using the STD method, it increased much more than when using the SKL algorithm. In addition, we ran the methods on an increasing number of TIRPs to detect, demonstrating a significant increase in the runtime for the STD method but not for the SKL algorithm.

Our original motivation was fast detection of multiple TIRPs to support efficient classification of new entities, represented as a series of symbolic intervals, given a large set of TIRPs that were discovered by a previous process as potential features for classification of such entities. Indeed, we have successfully used the SKL algorithm for precisely this objective [Moskovitch and Shahar, 2013].

However, a more general intriguing use for the new SKL algorithm is to use it for quickly answering a series of TIRP-based temporal queries referred to a large database of entities, which are represented as a series of raw time-stamped data. For example, a typical objective might be to classify multiple entities within such a time-oriented database; another one might be to simply select several of the entities for additional processing and/or display. Examples of such a use include selection and/or classification of longitudinal patient records for eligibility to a clinical trial or for a new clinical research; selection of financial records suspected for fraud; analysis of information-security records of devices that might have been infected by malware; etc. In all of these case, the goal is to match, to the set of entities represented as a set of symbolic time intervals, a series of temporal-constraints criteria, i.e., temporal queries, represented as complex patterns of symbolic time intervals and the relations amongst them; i.e., a set of TIRPs. The intention is to return for each entity, all of the instances of TIRPs that hold within it, including *when*, and *how many*. (The importance of completeness in finding *all* of the TIRPs was explained when presenting the KarmaLego algorithm).

When expecting the need for such a general temporal-query case, our intention is to first prepare for such queries by applying a temporal-abstraction process to all of the entities, in an offline manner, thus creating a series of symbolic intervals that represent each entity. Then, when given a series of temporal queries to the database, represented as a set of TIRPs, we intend to apply the SKL algorithm to all of the entities. Thus, we will quickly and efficiently determine precisely *which* (if any) TIRPs exist in each single entity, and if so, how many instances, and where.

Acknowledgements. The authors wish to thank Marion Verduijn for sharing the ICU12h dataset, and Prof. Avi Porath from the Soroka Academic Medical Center for his assistance regarding the diabetes data set. For insightful discussions about time intervals mining and classification using TIRPs our thanks to Christos Faloutsos, Christian Freksa, Fabian Moerchen, Dhaval Patel and Iyad Batel. This work was supported in part by grants from Deutsche Telekom Laboratories, and the HP labs Innovation Research Program.

R. Moskovitch, Y. Shahar. Fast Detection of Time Intervals Related Patterns. Technical Report 11/14,,Ben Gurion University, Beer Sheva, Israel, 2014.

References

J. F. Allen. Maintaining knowledge about temporal intervals, Communications of the ACM, 26(11): 832-843, 1983.

J. Ayres, J. Gehrke, T. Yiu, and J. Flannick, Proceedings SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, July 2002.

R. Azulay, R. Moskovitch, D. Stopel, M. Verduijn, E. de Jonge, and Y. Shahar, Temporal Discretization of medical time series - A comparative study, IDAMAP 2007, Amsterdam, The Netherlands, 2007.

I. Batal, H. Valizadegan, G. Cooper and M. Hauskrecht, A Temporal Pattern Mining Approach for Classifying Electronic Health Record Data. ACM Transaction on Intelligent Systems and Technology (ACM TIST), Special Issue on Health Informatics, 2012a.

I. Batal, D. Fradkin, J. Harrison, F. Moerchen, and M. Hauskrecht, Mining Recent Temporal Patterns for Event Detection in Multivariate Time Series Data, Proceedings of *Knowledge Discovery and Data Mining (KDD)*, Beijing, China, 2012b.

C. Freksa, Temporal reasoning based on semi-intervals, Artificial Intelligence, vol. 54, 1, 1992.

F. Höppner, Learning Temporal Rules from State Sequences, Proceedings of WLTSD, 2001.

F. Höppner, Time series abstraction methods - A Survey Workshop on Knowledge Discovery in Databases, Dortmund, 2002

B. Hu, Y. Chen, and E. Keogh, Time Series Classification under More Realistic Assumptions, Proceedings of SIAM Data Mining, 2013.

V. R. Jakkula, D. J. Cook, Detecting Anomalous Sensor Events in Smart Home Data for Enhancing the Living Experience, Artificial Intelligence and Smarter Living'11, 2011

P. S. Kam and A. W. C. Fu, Discovering temporal patterns for interval based events, In Proceedings DaWaK-00, 2000.

N. Lavrac, E. Keravnou and B. Zupan, Intelligent Data Analysis in Medicine, White Paper. Slovenia: Faculty of Computer Science and Information Sciences, University of Ljubljana, 2000.

J. Lin, E. Keogh, S. Lonardi, and B. Chiu, A Symbolic Representation of Time Series with Implications for Streaming Algorithms, In 8th ACM SIGMOD DMKD workshop, 2003.

F. Mörchen, and A. Ultsch, Optimizing Time Series Discretization for Knowledge Discovery, In Proceeding of KDD05, 2005.

F. Mörchen, Algorithms for Time Series Knowledge Mining, Proceedings of KDD, 2006.

F. Moerchen, A better tool than Allen's relations for expressing temporal knowledge in interval data, Workshop on Temporal Data Mining, 2006.

F. Moerchen, and D. Fradkin, Robust mining of time intervals with semi-interval partial order patterns, Proceedings of SIAM Data Mining, 2010.

R. Moskovitch, D. Stopel, M. Verduijn, N. Peek, E. de Jonge, and Y. Shahar, Analysis of ICU Patients Using the Time Series Knowledge Mining Method, IDAMAP 2007, Amsterdam, The Netherlands, 2007.

R. Moskovitch, Y. Shahar, Medical Temporal-Knowledge Discovery via Temporal Abstraction, AMIA 2009, San Francisco, USA, 2009.

R. Moskovitch, N. Peek, Y. Shahar, Classification of ICU Patients via temporal abstraction and temporal patterns mining, IDAMAP, Verona, Italy, 2009.

R. Moskovitch, Y. Shahar. Classification of multivariate time series via temporal abstraction and time intervals mining. Technical Report, Department of Information Systems Engineering, Ben Gurion University, Beer Sheva, Israel, 2013.

R. Moskovitch, Y. Shahar, Fast time intervals mining using transitivity of temporal relations, Knowledge and Information Systems, In Press.

P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, Mining frequent arrangements of temporal intervals, Knowledge and Information Systems, 21 (2), 2009.

D. Patel, W. Hsu, M L Lee, Mining Relationships among Interval-based Events for Classification, Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008.

J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, Proc. 17th Int'l Conf. Data Eng. (ICDE '01), 2001.

L.R. Rabiner, A tutorial on Hidden Markov Models and selected applications in speech recognition, Proceedings of the IEEE, 77, 1989.

C Ratanamahatana, E. J. Keogh, Three Myths about Dynamic Time Warping Data Mining, Proceedings of Siam Data Mining, 2005.

R. Moskovitch, Y. Shahar. Fast Detection of Time Intervals Related Patterns. Technical Report 11/14,,Ben Gurion University, Beer Sheva, Israel, 2014.

J. Roddick and M. Spiliopoulou, A survey of temporal knowledge discovery paradigms and methods. IEEE Transactions on Knowledge and Data Engineering, 4(14), 2002.

L. Sacchi, C. Larizza, C. Combi, and R. Bellazi, Data mining with temporal abstractions: learning rules from time series. Data Mining and Knowledge Discovery, (15), 2007.

G. Salton, A. Wong, and C.S. Yang, A vector space model for automatic indexing, Communications of the ACM, 18, 1975.

Y. Shahar, A framework for knowledge-based temporal abstraction, Artificial Intelligence, 90(1-2) 1997.

Y. Shahar, Dynamic temporal interpretation contexts for temporal abstraction, Annals of Mathematics and Artificial Intelligence 22 (1-2), 1998.

Y. Shahar, Knowledge-based temporal interpolation, Journal of Experimental and Theoretical Artificial Intelligence 11, 1999.

Y. Shahar, H. Chen, D. Stites, L. Basso, H. Kaizer, D. Wilson, and M.A. Musen, Semiautomated acquisition of clinical temporal-abstraction knowledge, Journal of the American Medical Informatics Association 6(6), 494-511, 1999.

M. Verduijn, L. Sacchi, N. Peek, R. Bellazi, E. de Jonge,B. de Mol. Temporal abstraction for feature extraction: A comparative case study in prediction from intensive care monitoring data, In Artificial Intelligence in Medicine, 41, 112, 2007.

R. Villafane, K. Hua, D Tran, and B. Maulik, Knowledge discovery from time series of interval events, Journal of Intelligent Information Systems, 15(1), 2000.

E. Winarko and J. Roddick. Armada - an algorithm for discovering richer relative temporal association rules from interval-based data, Data and Knowledge Engineering, 1(63), 2007.

S. Wu, Y. Chen, Mining Non-ambiguous Temporal Patterns for Interval-Based Events, IEEE Transactions on Knowledge and Data Engineering, 19 (6), 2007.