

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/273514315>

Finding the longest common sub-pattern in sequences of temporal intervals

Article in *Data Mining and Knowledge Discovery* · February 2015

DOI: 10.1007/s10618-015-0404-3

CITATIONS

9

READS

133

2 authors:



[Orestis Kostakis](#)

Microsoft

13 PUBLICATIONS 317 CITATIONS

[SEE PROFILE](#)



[Panagiotis Papapetrou](#)

Stockholm University

119 PUBLICATIONS 1,228 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Machine Learning-based Triage of Digital Evidence Sources [View project](#)



AI for Education [View project](#)

Finding the longest common sub-pattern in sequences of temporal intervals

Orestis Kostakis · Panagiotis Papapetrou

Received: 2 March 2014 / Accepted: 4 February 2015 / Published online: 19 February 2015
© The Author(s) 2015

Abstract We study the problem of finding the longest common sub-pattern (LCSP) shared by two sequences of temporal intervals. In particular we are interested in finding the LCSP of the corresponding arrangements. Arrangements of temporal intervals are a powerful way to encode multiple concurrent labeled events that have a time duration. Discovering commonalities among such arrangements is useful for a wide range of scientific fields and applications, as it can be seen by the number and diversity of the datasets we use in our experiments. In this paper, we define the problem of LCSP and prove that it is NP-complete by demonstrating a connection between graphs and arrangements of temporal intervals. This connection leads to a series of interesting open problems. In addition, we provide an exact algorithm to solve the LCSP problem, and also propose and experiment with three polynomial time and space under-approximation techniques. Finally, we introduce two upper bounds for LCSP and study their suitability for speeding up 1-NN search. Experiments are performed on seven datasets taken from a wide range of real application domains, plus two synthetic datasets. Lastly, we describe several application cases that demonstrate the need and suitability of LCSP.

Keywords Temporal intervals · Longest common sub-pattern · Event-interval sequences

Responsible editors: Toon Calders, Floriana Esposito, Eyke Hüllermeier, Rosa Meo.

O. Kostakis
Aalto University, Konemiehentie 2, 02150 Espoo, Finland
e-mail: orestis.kostakis@aalto.fi

P. Papapetrou (✉)
Department of Computer and Systems Sciences, Stockholm University,
Forum 100, 164 40 Kista, Sweden
e-mail: panagiotis@dsv.su.se

1 Introduction

Sequences of temporal intervals, also known as *event-interval sequences*, have recently attracted the attention of both the databases and data mining communities. Such sequences are ubiquitous and their main characteristic is that they consist of events that are not necessarily instantaneous but may have a time duration. Sequences of this type appear in several application domains, such as sign language (Papapetrou et al. 2009), medicine (Kosara and Miksch 2001), geo-informatics (Pissinou et al. 2001), cognitive science (Berendt 1996), linguistics (Bergen and Chang 2005), and music informatics (Pachet et al. 1996).

The main advantage of event-interval sequences is that they are a generalization of traditional event sequences, since they do not restrict events to be instantaneous but they allow them to have a time duration. Hence, they are constructed by events that may exhibit different temporal relations. Formally, event-interval sequences can be considered as an ordered multiset of events characterized by a label, a start, and an end time value. Event labels are allowed to occur multiple times within the same sequence, since such property is required in several application domains, such as sign language (Papapetrou et al. 2009). An example of an event-interval sequence containing five labeled events, i.e., *A* (occurring twice), *B*, *C*, and *D*, is shown in Fig. 1. Furthermore, multiple events of the same label can be active simultaneously. An example of such case would be the representation of recursion, when monitoring the execution of a computer program. Another example would be the presence of groups or classes of equivalent sensors, in sensor networks, where it is unnecessary or undesired to identify the specific sensor id.

In sign language, for instance, a sentence is constructed by events that may correspond to grammatical and syntactic expressions, or various hand and facial gestures. Such events have a time duration and may also occur concurrently, hence building sequences of labeled temporal intervals. Examples of sign language event labels include “Wh-Word”, “Lowered eyebrows”, or “Rapid head shake”. In Fig. 2 we can see an example of a *Wh-question* (question starting with a word prefixed by ‘Wh’). It can be observed that event intervals may occur concurrently exhibiting several temporal relations between them, while the same event label, i.e., “Wh-Word”, may occur multiple times within the same sequence. As another example, consider a medical database populated by records of patients who follow a series of medication and tests for some time period. In such setting, an event corresponds to a prescribed drug or a

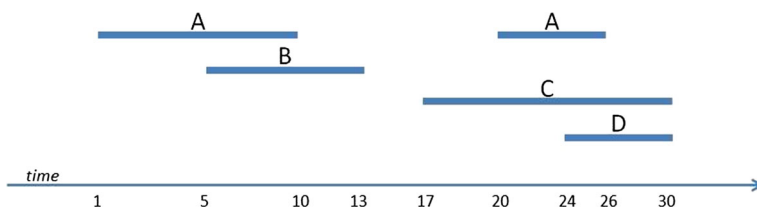


Fig. 1 Example of a sequence of five events, each occurring over a time interval. The same event label may occur multiple times in the sequence, while several temporal relations may occur between the event intervals

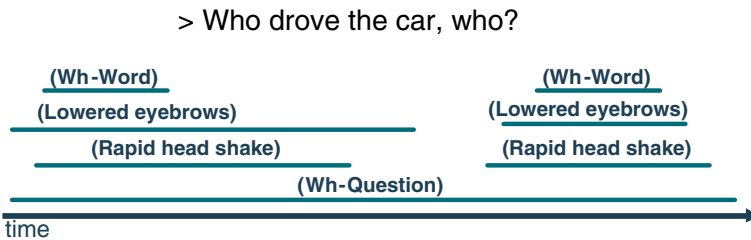


Fig. 2 Example of a sign language event-interval sequence (Papapetrou et al. 2009). The sequence represents the phrase “Who drove the car, who?” expressed using sign language notation. It can be seen that event labels may repeat throughout the sequence while event intervals may occur concurrently and exhibit different types of temporal relations

medical test. Similar to the previous example, it is again apparent that events can also occur concurrently and over a time interval.

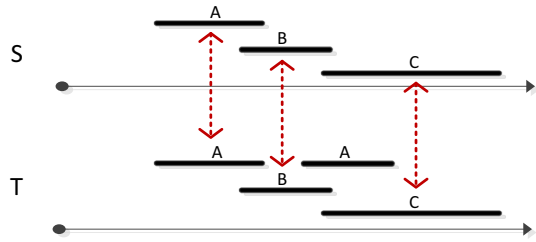
Recent work has focused on event-interval sequences, but has mainly concentrated on mining frequent patterns and association rules (Kam and Fu 2000; Ale and Rossi 2000; Papapetrou et al. 2009; Mörchen 2007), mining semi-interval partial orders (Mörchen and Fradkin 2010), or discovering relationships for classification (Patel et al. 2008). Despite that, it is surprising that other important problems such as similarity search and matching have received very limited attention.

Recently, a family of methods has been proposed for similarity search in event-interval sequence repositories (Kostakis et al. 2011; Kotsifakos et al. 2013). Their key approach is to use a simplified representation for each event-interval sequence without losing crucial temporal information about the events. Nonetheless, all existing methods have been designed for *full sequence matching*. In other words, they attempt to quantify the dissimilarity of two given event-interval sequences. The difference between full-sequence matching and longest common sub-pattern (LCSP) is the same as that between computing the string Edit Distance and the longest common subsequence (LCS) of two strings. That is, full-sequence matching “forces” all elements in one sequence to match with at least one element in the other sequence. In other words, it penalizes any element mismatch between the two sequences. On the other hand, LCSS allows for gaps in the alignment. Hence, it is more elastic to noise since outliers cannot distort the similarity as they are not matched.

There are many application domains where it is desirable to search for “commonalities” between two sequences, where the main task is to extract segments of the two sequences that are similar to each other. Such task is highly applicable in biology, known as *local alignment* (Smith and Waterman 1981), as well as in time series, e.g., LCSS (Paterson and Dancik 1994). In graphs, the problem is known as the maximum common subgraph (MCS) isomorphism. In practice, LCSP can be used for several tasks: (i) given two e-sequences S, T , determine the largest pattern of intervals that appears in both, (ii) given a query e-sequence pattern q , and a larger e-sequence S , determine the extent to which q exists in S . An alternative view on the latter is that of approximate sub-sequence querying for sequences of event-intervals.

In the case of event-interval sequences, identifying such commonalities may be highly beneficial in various use-case scenarios. Consider the case of *sign language*

Fig. 3 Example of the LCSP between two event-interval sequences S and T . LCSP identifies the longest common sub-pattern (LCSP) between the two sequences by effectively allowing skipping events in the matching



classification. Using a set of known profiles that characterize the nature or structure of various sequence samples in a database, and given a query sequence, one could compare the unknown sequence to those profiles to determine its class. For example, suppose we have a set of common profile patterns for various sign language expressions, such as “Wh-questions” and “Negations”. A very common profile, for instance, for a “Wh-question” is a “Wh-word” overlapping with “Lowered eyebrows” and a “Rapid head shake”. This pattern can also be seen in Fig. 2. Given a new, unclassified sequence sample, existing profiles could be used to determine the class of the new sample by identifying commonalities between the profiles and the new sequence.

In this paper, we study the problem of identifying the LCSP between two event-interval sequences. In other words, our goal is to identify the longest commonality between the two sequences, expressed as patterns of event intervals sharing the same temporal relations. An example, of LCSP is shown in Fig. 3. The two event-interval sequences S and T share the same “sub-pattern” consisting of events A , B , and C . It becomes apparent that such pattern cannot be identified by existing full sequence matching algorithms, as they require each event interval from one sequence to be mapped to at least one event interval in the other sequence.

In our model, we are not concerned with the actual duration of the intervals nor with the time separating any intervals. Instead we focus on the actual combination of the intervals and their relations. This makes our measure robust to any time warping. So, the two sequences $\{(A, 1, 2), (B, 3, 4)\}$ and $\{(A, 10, 20), (B, 50, 100)\}$ are considered the same (each triple denotes respectively the label, start and end time of an interval). The motivation and rational for this is that we should expect from people who practice Sign Language, when repeating a phrase, to consume different amounts of time for each word and the whole utterance between different attempts; similar to when pronouncing a long sentence in spoken language. In the same manner, we assume that in robot sensor data a high-level description of a situation is derived from the combination of underlying events. For example, the gripping mechanism was enabled throughout the whole time that the robot’s wheels were active, so the object was transferred successfully to the target location, in contrast to releasing at any time point half-way through which would indicate a drop.

The reader may wonder whether this problem could be simply solved by converting event-interval sequences to discrete event sequences, hence mapping the LCSP problem to the longest common subsequence (LCS) problem (Cormen et al. 2001). It has been demonstrated and argued in several existing works (Kostakis et al. 2011; Papapetrou et al. 2009) that event-interval sequences cannot be directly mapped to discrete event sequences without loss of temporal information; especially when temporal

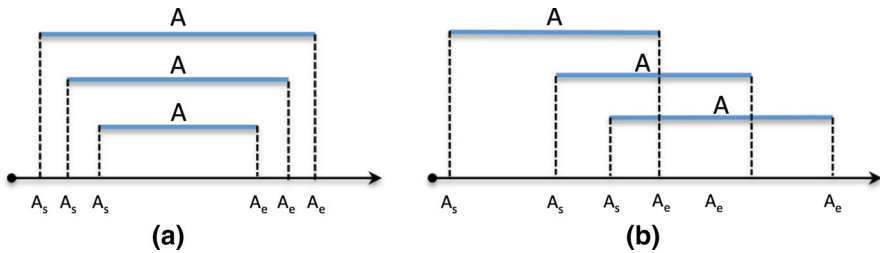


Fig. 4 An example where mapping event-interval sequences to discrete event sequences may cause ambiguities. By adding more intervals while maintaining the same structure in both sequences, we demonstrate that reducing LCSP to string matching produces arbitrarily bad results

relations such as *overlaps* or *contains* are allowed between event intervals sharing the same label. This shortfall results in side-effects for other data mining tasks; as we demonstrate experimentally on seven real datasets in Sect. 7.3, LCSP can achieve much better performance both in terms of classification accuracy and clustering purity.

Next, we demonstrate that the aforementioned mapping to discrete sequences can produce arbitrarily bad scores irrespective of the similarity measure used. Consider the two examples shown in Fig. 4, where each event-interval sequence consists of three intervals with the same label. In the first case (Fig. 4a), each event interval is fully contained within the other (in terms of duration), while in the second case (Fig. 4b) each event interval overlaps with all the previous. Obviously, the mapping for both sequences is the same, i.e., $\{A_s, A_s, A_s, A_e, A_e, A_e\}$. This suggests that traditional methods for discrete event sequences may fail to capture the inherent temporal structure of such sequences, and more important, they may produce arbitrarily bad results especially as the number of event labels increases. In the same example, consider the case where we have infinitely many event intervals: for Fig. 4a each new event interval is contained within the others, while for Fig. 4b each new event interval overlaps with all the previous. Both event-interval sequences would be mapped to the same two discrete event sequences and hence any string matching algorithm would match them fully and their LCS would be of length $2 \times |S| = 2 \times |T|$, whilst in reality S and T share no common temporal relation between their event intervals, hence their LCS is just of length 1, i.e., event interval A .

The main *contributions* of this paper are summarized as follows:

- We formally define the problem of finding the LCSP between a pair of arrangements of temporal intervals.
- We prove that the LCSP problem belongs to the complexity class of NP -hard problems, by showing that Clique can be reduced to it under a log-space reduction. We achieve this by establishing that arrangements can be used to encode graphs.
- We prove $LCSP \in NP$ by showing a reduction to Max Clique.
- We demonstrate an exact algorithm and prove of its correctness.
- We describe a framework for the problem variation of inexact LCSP.
- We propose three policies for under-approximating (approximating from below) hard instances of LCSP. We benchmark them in terms of running time and accuracy (tightness).

- We propose two upper-bounds for LCSP and study their tightness and 1-NN pruning power.
- We experiment on seven real datasets taken from various domains, including sign language, medicine, human motion, and sensor networks, and two additional synthetic datasets.

The remainder of this paper is organised as follows: in Sect. 2 we summarize the related work, in Sect. 3 we provide the necessary definitions and the problem formulation, then in Sect. 4 we present the LCSP problem, show that it is NP-hard, and present an exact algorithm for solving it. Then, in Sect. 5 we present three approximations for solving the problem and in Sect. 6 we propose two upper bounds for computing LCSP. Next, in Sect. 7 we present our experimental evaluation and in Sect. 8 we provide several motivating use cases for the applicability and suitability of LCSP. Finally, Sect. 9 concludes the paper and presents directions for future research.

2 Related work

The vast majority of related research on temporal interval sequences has so far been focusing merely on frequent pattern and association rule mining, as opposed to similarity matching, which is the main focus of this paper. The simplest formulation is inspired by the idea of itemset mining and mainly considers events to be time intervals. Hence, the task at hand is to discover frequently occurring patterns of intervals in databases, irrespective of labels (Lin 2003; Villafane et al. 2000). Similar approaches (Giannotti et al. 2006) focus on extracting temporally annotated sequential patterns, where transitions from one event to another have a time duration. In such scenario, intervals correspond to the time differences between the offset (end time) and onset (start time) of an event.

A graph-based approach (Hwang et al. 2004) represents each temporal pattern by considering only two types of relations between event-intervals (*follow* and *overlap*), and illustrate examples from various application domains where discovery of temporal patterns can be applied to support crucial business decision-making. Ale and Rossi (2000) approach the concept of a temporal interval by modeling the lifetime of an item as the time between its first and last occurrence.

Another family of methods on temporal intervals are those that consider sequences of labeled temporal intervals, and extract temporal patterns in such sequences. A large variety of Apriori-based techniques (Kam and Fu 2000; Abraham and Roddick 1999; Chen and Petrounias 1999; Höppner 2001; Höppner and Klawonn 2001; Mooney and Roddick 2004; Laxman et al. 2007) for finding temporal patterns, episodes, and association rules on interval-based event sequences have been proposed. More sophisticated methods on pattern mining in sequences of temporal intervals employ enumeration trees for candidate generation and pruning (similar to those for traditional itemset and sequential pattern mining). Significant speedups are achieved by BFS-based and DFS-based enumeration on these trees (Winarko and Roddick 2007; Papapetrou et al. 2009), by reducing the inherent exponential complexity of the mining problem. A non-ambiguous temporal interval representation is presented in (Wu and Chen 2007) that considers start and end points of event intervals, and converts them to a sequential

representation. Nonetheless, the alphabet size (number of event labels) as well as the overall complexity of the mining process is increased. In addition, temporal relations such as *overlaps* or *during* between the same event label still cannot be distinguished. Furthermore, efficient methods have been proposed on mining partial orders of semi-intervals (Mörchen and Fradkin 2010) as well as closed patterns of interval-based events has been proposed (Chen et al. 2011).

Recent work on *margin-closed* patterns (Fradkin and Moerchen 2010) focuses on significantly reducing the number of reported patterns by favoring longer patterns and suppressing shorter patterns with similar frequencies. A unifying view of temporal concepts and data models has been formulated in (Mörchen 2007) to enable categorization of existing approaches to unsupervised pattern mining from symbolic temporal data; time point-based methods and interval-based methods as well as univariate and multivariate methods are considered. In addition, an encoding scheme for compressing sequence data with sequential patterns has been proposed (Lam et al. 2014) for solving the problem of compressing sequential patterns from a sequence database.

All the aforementioned approaches and formulations are beyond the scope of this paper, since their objectives and formulations are orthogonal to ours. To the best of our knowledge, the only existing principled methods for assessing the similarity of sequences of temporal intervals are Artemis (Kostakis et al. 2011) and IBSM (Kotsifakos et al. 2013). The first one is based on the fraction of common temporal relations between the sequences, without taking into account the actual time durations, while the second performs a vector-based representation of each point in time and maps the problem to a Euclidean distance computation between ordered sets of vectors. Additionally, a baseline approach, called DTW-based, is presented in Kostakis et al. (2011). However, this method employs a vector-based representation of event-interval sequences and, due to its construction, fails to consider any pair-wise temporal relation. It should be noted that both methods assess the similarity of two sequences of temporal intervals by performing *full sequence matching*; that is each element in one sequence should be matched to an element in the other sequence. This objective is however orthogonal to our objective in this paper, since we are solving a different problem: how to find the longest common subpattern shared by two sequences of temporal intervals. Hence, we are the first to formulate the problem of LCSP and apply it for assessing the similarity of event-interval sequences.

3 Background

Let $\Sigma = \{E_1, \dots, E_m\}$ be an alphabet of m event labels. An event that occurs over a time interval defines an *event interval* and an ordered multiset of event intervals defines an *event-interval sequence*. Next, we provide a more formal definition for these two concepts.

Definition 1 (Event interval) An *event interval* is defined as a triple $S = (E, t_{start}, t_{end})$, where $S.E \in \Sigma$ and $S.t_{start}, S.t_{end}$ correspond to the start and end time of S , respectively. $S.t_{start} \leq S.t_{end}$, where the equality holds when the event is instantaneous. For ease, we also denote $S.E$ as E_S .


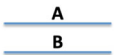
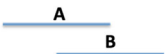
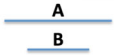



	<i>A meets B</i>
	<i>A is equal to B</i>
	<i>A overlaps with B</i>
	<i>A during B</i>
	<i>A finishes with B</i>
	<i>A starts with B</i>
	<i>A before B</i>

Fig. 5 The seven temporal relations between two event-intervals that are considered in this paper

Definition 2 (e-sequence) A sequence of temporal intervals, or *event-interval sequence*, or *e-sequence*, $\mathcal{S} = \{S_1, \dots, S_n\}$ is an ordered list of n triples (i.e., event intervals) that may contain duplicates. The temporal order of the event intervals in \mathcal{S} is ascending based on their start time and in the case of ties it is descending based on their end time. If ties still exist, the event intervals are sorted lexicographically.

An example of an e-sequence is shown in Fig. 1. Using the above definitions, this e-sequence is represented as follows:

$$\mathcal{S} = \{(A, 1, 10), (B, 5, 13), (C, 17, 30), (A, 20, 26), (D, 24, 30)\}.$$

It becomes apparent that in an e-sequence there exist temporal relations between the event intervals. Based on Allen's model for temporal interval relations (Allen 1983; Allen and Ferguson 1994), given two event intervals A and B , we consider the following seven relations (shown in Fig. 5): *before*(A, B), *meets*(A, B), *equal*(A, B), *overlapsWith*(A, B), *during*(A, B), *startsWith*(A, B), *finishesWith*(A, B).

More details about these relations can be found in Papapetrou et al. (2009). Let $\mathcal{I} = \{r_1, \dots, r_{|\mathcal{I}|}\}$ denote the set of all legal temporal relations that can exist between any pair of event-intervals. For our setting, we have $|\mathcal{I}| = 7$ with

$$\mathcal{I} = \{\textit{meets}, \textit{is equal to}, \textit{overlaps with}, \textit{during}, \textit{finishes with}, \textit{starts with}, \textit{before}\}.$$

In several applications, one may be interested not so much in the absolute time values of the start and end points of event-intervals but rather in the types of temporal relations between them. Hence, a simplified representation may be used, which is called *arrangement* (Papapetrou et al. 2009).



Fig. 6 Example of an arrangement of length three and size three. The arrangement (on the *right*) is extracted from the e-sequence (on the *left*) by removing the time stamps and by taking into account only the temporal relations between the event intervals

Definition 3 (Arrangement) An arrangement $\mathcal{A} = \{\mathcal{E}, \mathcal{R}\}$ of length n consists of a sequence of event labels \mathcal{E} , with $|\mathcal{E}| = n$, and a set of relations $\mathcal{R} = \{R(E_1, E_2), R(E_1, E_3), \dots, R(E_{n-1}, E_n)\}$, where each $R(E_i, E_j) \in \mathcal{I}$ denotes the temporal relation between E_i and E_j , for $i = 1, \dots, n-1$ and $j = i+1, \dots, n$.

Intuitively, an arrangement can be seen as a summary of an e-sequence with respect to the event labels and pair-wise event relations that are present in the e-sequence. An arrangement can be generated from an e-sequence by maintaining the interval relation structure and disregarding the exact values of the start and end points. Finally, the *length* of an arrangement \mathcal{A} is defined as the number of its event intervals (denoted as $|\mathcal{A}|$), while its *size* is the number of temporal relations in \mathcal{A} .

An example of an arrangement is given in Fig. 6, where on the left hand side we can see the original e-sequence and on the right hand side the corresponding arrangement representation. Observe that the time stamps are dropped and only the relation types are maintained. The length of this arrangement is 3 (three event labels: A, B, and C), while its size is also 3 (three temporal relations: A overlaps with B, A before C, and B before C).

For the remainder of this paper we essentially focus on the “arrangement” representation of e-sequences since we are not particularly interested in absolute values of event durations but only on the relations between the events; for reasons argued in the Introduction.

4 Longest common sub-pattern

In this section we formulate and study the problem of finding the LCSP between a pair of arrangements of temporal intervals. We formally define LCSP and prove that finding it is NP -complete. We also describe a framework for finding the approximate LCSP of two arrangements; a relaxation of LCSP where a certain threshold of relations disagreement is allowed. Furthermore, we present an exact algorithm to retrieve the LCSP of pairs of arrangements, which is based on dynamic programming (DP).

Definition 4 (Longest common sub-pattern) Given two arrangements, \mathcal{A} and \mathcal{B} , their LCSP is the maximum sequence of intervals

$$S_{LCSP} = \{S_{LCSP_1}, S_{LCSP_2}, \dots, S_{LCSP_k}\},$$

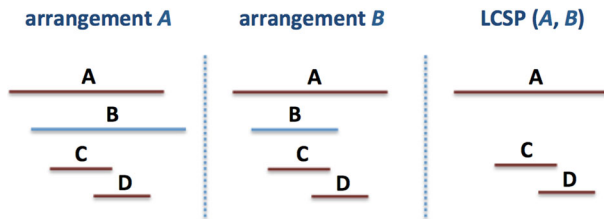


Fig. 7 The LCSP of the two arrangements is the sub-arrangement formed by the intervals with labels A, C and D

such that there exist two sets of event-intervals

$$\{S_{a1}, S_{a2}, \dots, S_{ak}\} \text{ in } \mathcal{A}, \text{ and } \{S_{b1}, S_{b2}, \dots, S_{bk}\} \text{ in } \mathcal{B}$$

and $\forall i, j$ s.t. $1 \leq i \leq j \leq k$ the following two equations hold:

$$E_{SLCSPi} = E_{S_{ai}} = E_{S_{bi}}. \quad (1)$$

$$R(E_{SLCSPi}, E_{SLCSPj}) = R(E_{S_{ai}}, E_{S_{aj}}) = R(E_{S_{bi}}, E_{S_{bj}}). \quad (2)$$

In other words, given a pair of arrangements \mathcal{A} and \mathcal{B} , the LCSP of that pair is an arrangement whose event intervals are a subset of those present in both \mathcal{A} and \mathcal{B} . For any pair of event intervals in the LCSP, the corresponding pairs of event intervals in \mathcal{A} and in \mathcal{B} have the same type of relation. For example, in Fig. 7, the LCSP of the two arrangements (on the left and in the middle) is the pattern formed by event intervals A, C, D (on the right). Interval B cannot be in a common sub-pattern together with interval A or D, since their relation is different in the two arrangements.

4.1 Complexity of LCSP

We prove that the LCSP problem is NP-complete by demonstrating its relation to the Clique problem. In the decision version of Clique, given as input a graph G and an integer k the goal is to determine whether G contains a k -clique. In the optimization version of Clique, *Max Clique*, the goal is to find the maximum clique in the input graph G . Similarly, the decision version of LCSP, namely CSP, is to determine whether there exists a sub-pattern of size k in both of the input arrangements. In this section, we show a reduction from Clique to CSP. It is a well known fact that given an algorithm for the decision version of a problem, the result of the optimization version can be retrieved using a logarithmic number of look-ups.

Lemma 1 Any undirected graph $G = (V_G, E_G)$ can be encoded as an arrangement of temporal intervals.

Proof In the transformation of graphs into arrangements, each vertex $u_i \in V_G$ corresponds to a time point t_i ; the correspondence between vertices and time points is not important as long as it is consistent throughout the procedure. For every edge

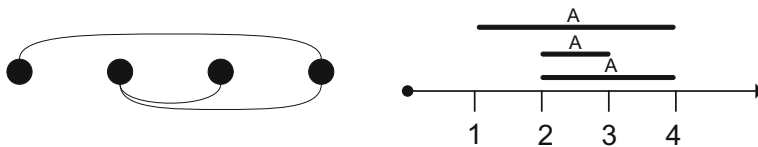


Fig. 8 An example of how a graph can be represented by an e-sequence. The list of edges is: $\{(1,4),(2,4),(2,3)\}$ while the e-sequence is encoded as: $\{(A,1,4),(A,2,4),(A,2,3)\}$

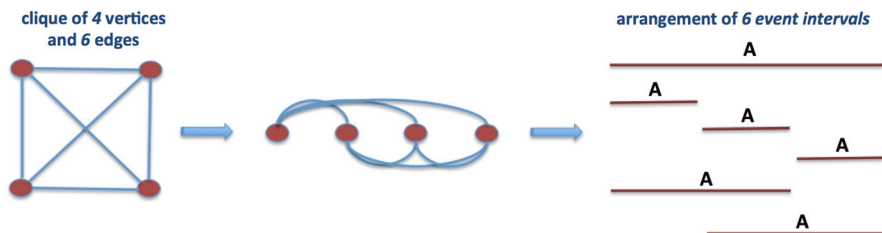


Fig. 9 An example of how to convert a 4-clique to an arrangement. All nodes are given the same label, and each edge corresponds to an event-interval

$e = (u_i, u_j) \in E_G$ we create interval $S = (a', t_i, t_j)$ (if $t_i < t_j$, else $S = (a', t_j, t_i)$). There is no reason to specifically select 'a' as a label, but it is important that all labels are the same. Unconnected vertices correspond to time points in which no intervals start or end. Edges of the form $(u_i, u_i) \in E_G$ can be conserved by creating instantaneous events of the form (a', t_i, t_i) . \square

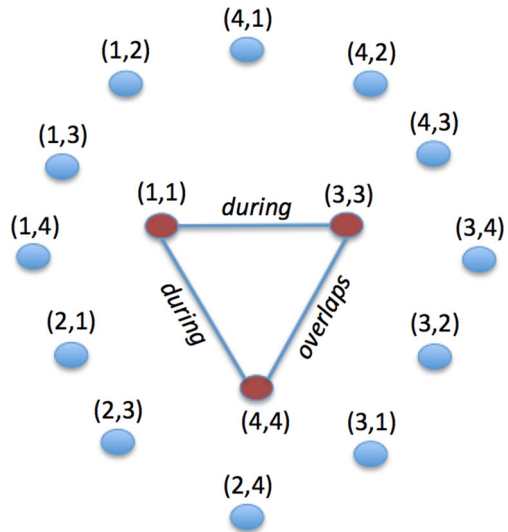
An example of how a graph can be encoded as a e-sequence is shown in Fig. 8. Note that the reverse procedure, from e-sequences to graphs (with time points still corresponding to vertices), would create multigraphs with labeled edges.

Theorem 1 *Clique can be reduced to CSP.*

Proof The graph G is transformed into an arrangement \mathcal{A}_G as described above. Given parameter k , we create a second arrangement \mathcal{A}_k that corresponds to a clique of size k . An example of a 4-clique converted to an arrangement is depicted in Fig. 9. This is easily achieved by creating all possible $k(k-1)/2$ intervals of the form $S = (a, t_i, t_j)$, with $1 \leq i \leq k-1, i < j \leq k$. The result of CSP determines the result for Clique. The whole arrangement \mathcal{A}_k is found in \mathcal{A}_G , or equivalently there exists a CSP between \mathcal{A}_G and \mathcal{A}_k of size equal to $k(k-1)/2$, if and only if the graph contains a clique of size k . Proving the last statement:

- (\Leftarrow) If graph G has a clique of size k , then there is a CSP of size $k(k-1)/2$: If graph G has a clique of size k , then there exist k vertices that are fully connected. Suppose the vertices of the clique are $V_{clique} = \{u_{c1}, \dots, u_{ck}\}$. The reduction would create \mathcal{A}_G containing, among others, all $k(k-1)/2$ intervals of the form $(a, t_{u_{ci}}, t_{u_{cj}})$, with $1 \leq i \leq k-1, i < j \leq k$. The reduction would also create \mathcal{A}_k , which contains exactly $k(k-1)/2$ intervals in a pattern identical to that formed by the intervals corresponding to the clique. Thus, there would be a CPS of size $k(k-1)/2$.

Fig. 10 An example of the Cartesian graph for the two arrangements given in Fig. 7. Observe that their LCSP is a 3-clique. Note that in both e-sequences the indices of interval labels A, B, C, D are 1, 2, 3, 4, respectively



- (\Rightarrow) If there exists a CSP of size $k(k-1)/2$, then the graph G has a clique of size k : If there exists a CSP of size $k(k-1)/2$ then there exists a set of intervals in \mathcal{A}_G which create a pattern identical to that of \mathcal{A}_k . That means there exist $k(k-1)/2$ intervals in \mathcal{A}_G of the form $S = (a, t_i, t_j)$ for all $1 \leq i \leq k-1, i < j \leq k$. Thus, k vertices exist in G connected by edges (u_i, u_j) for all $1 \leq i \leq k-1, i < j \leq k$. Equivalently, k nodes in G are fully connected. \square

Theorem 2 *LCSP reduces to Max Clique.*

Proof This is done by transforming an LCSP instance of two e-sequences A, B into a Max Clique instance of a single graph G_{AB} of order $|V_{G_{AB}}| = nm$, where $n = |A|$, $m = |B|$. Given two e-sequences A, B , for each pair of intervals $i \in A, j \in B$, we create a node with label (i, j) . In other words, the set of vertex labels of G_{AB} is the Cartesian product of the sets of interval indices of the two e-sequences. We add edges between the nodes with labels (i, j) and (k, l) only if $E_{A_i} = E_{B_j}, E_{A_k} = E_{B_l}$ and $R(E_{A_i}, E_{A_k}) = R(E_{B_j}, E_{B_l})$. Vertices in G_{AB} that correspond to pairs intervals with different labels would be disconnected. We call this graph the *Cartesian graph* of two e-sequences. An example of this graph and how to map two e-sequences to their Cartesian graph and detect their LCSP is given in Fig. 10.

It follows naturally from the definition of LCSP, that finding the LCSP of two e-sequences A, B is equivalent to finding the maximum clique in G_{AB} . Similarly, finding maximal cliques in G_{AB} corresponds to finding maximal common sub-patterns between A and B . \square

From the above reductions, it may appear as if LCSP is identical to finding the MCS of two graphs. However this is not the case. We may transform e-sequences into graphs in two ways. First, as described in Lemma 1, time-points correspond to vertices, and the intervals would correspond to edges. The issue in this case would

be that any two intervals, each one corresponding to an edge, may have a different interval relation (“meets”, etc). Hence, any two edges in a graph do not necessarily correspond to (for example) overlapping intervals. In other words, there exists crucial temporal information in e-sequences that should be preserved when representing them by some other structure. Furthermore, they would be multigraphs with labeled edges. The second transformation would consider each interval represented by a unique vertex and the relations would be denoted by the edge labels. In this case and based on the definition of LCSP, the problem would reduce to finding the Maximum Common Complete Subgraph. In other words, the common subgraph should be a clique.

4.2 LCSP with errors

Suppose we can relax the strict definition of the problem and allow a certain number of interval relations to not be identical. In other words, we are interested in finding a largest common sub-pattern where a certain threshold of error is allowed. This corresponds to finding sets of vertices in the Cartesian graph (see proof of Theorem 2 for definition) that would form a dense subgraph that is a few edges far from being a clique. More formally, the problem translates to finding quasi-cliques in the Cartesian graph; a quasi-clique is an induced sub-graph of k vertices with $\alpha \binom{k}{2}$ edges, $\alpha \in (0, 1]$, $k \leq |V_G|$. The value of α denotes the *density* of the quasi-clique. In the LCSP context, $1 - \alpha$ would denote the allowed error rate.

In this work, we do not study any further the problem of explicitly extracting an LCSP with errors, since the problem of extracting quasi-cliques was studied recently by (Tsourakakis et al. 2013; Jiang and Pei 2009; Liu and Wong 2008).

4.3 An exact algorithm for LCSP

We present an exact algorithm, based on DP, to retrieve the LCSP between pairs of arrangements. The algorithm constructs arrangements using pairs of intervals such that each interval in a pair corresponds to one interval from each sequence. Those constructed arrangements are a subset of all the maximal sub-arrangements. The final goal is to find the maximal sub-arrangement that has the largest size.

4.3.1 Computing LCSP

The main steps of the exact DP algorithm are depicted in Algorithm 1. The final state of the DP array for the two arrangements in Fig. 7 is depicted in Table 1.

The input of the algorithm comprises of two arrangements $\mathcal{A} = \{S_{A1}, \dots, S_{Am}\}$ and $\mathcal{B} = \{S_{B1}, \dots, S_{Bn}\}$. We denote by $LCSP(i, j)$, $1 \leq i \leq |\mathcal{A}|$, $1 \leq j \leq |\mathcal{B}|$, the set of maximal CSPs between $\{S_{A1}, \dots, S_{Ai}\}$ and $\{S_{B1}, \dots, S_{Bj}\}$, where each CSP must include an interval corresponding to S_{Ai} and S_{Bj} ; in other words A_i is matched to B_j . If $E_{SA_i} \neq E_{SB_j}$ then $LCSP(i, j) = \{\emptyset\}$, otherwise we identify the maximal sub-arrangement that spans the prefixes of the two arrangements and matches A_i is matched to B_j . The key intuition here is that we can infer $LCSP(i, j)$ if we know all previous $LCSP(p, q)$, with $1 \leq p < i$ and $1 \leq q < j$.

Table 1 Final view of the DP array of the exact LCSP algorithm on the instance of Fig. 7

	A	B	C	D
D	\emptyset	\emptyset	\emptyset	$[(1, 1), (3, 3), (4, 4)]$
C	\emptyset	\emptyset	$[(1, 1), (3, 3)], [(2, 2)(3, 3)]$	\emptyset
B	\emptyset	$[(2, 2)]$	\emptyset	\emptyset
A	$[(1, 1)]$	\emptyset	\emptyset	\emptyset

Unlike the case of the LCS for strings, in arrangements it is not sufficient to know only $LCS(i - 1, j - 1)$, $LCS(i - 1, j)$ and $LCS(i, j - 1)$. Furthermore, given any element from $LCS(p, q)$, it is not sufficient to simply append the new interval. Actually, it is not even correct. The reason is that all other previous intervals may have different relations with S_{Ai} and S_{Bj} and not necessarily a *before* relation. For example, in Fig. 7 the interval labeled as “D” has a different relation with the interval labeled as “B” in the two arrangements respectively, although “C” is the previous of “D”. Thus, it is not sufficient to check just the relation with the last interval. Instead, S_{Ai} and S_{Bj} must be checked against all intervals of the sub-solution and keep only those that have the same relation (thus, it is possible to have $|LCS(p, q)| \geq |LCS(i, j)|$); this is performed using the \otimes operation that we explain below. We will abuse the notation $LCS(i, j)$ to denote alternately both any or all of the maximal sub-arrangements produced at point (i, j) .

Critical to our algorithm is the \otimes operation that is applied between a sub-arrangement and a pair of intervals. In particular, $LCS(p, q) \otimes (i, j)$ denotes the arrangement that occurs from the interval corresponding to S_{Ai} and S_{Bj} , and the event-intervals in $LCS(p, q)$ whose correspondents have the same relations with intervals S_{Ai} and S_{Bj} in \mathcal{A} and \mathcal{B} respectively. For example, in Fig. 7, suppose that $\{B, C\}$ has been discovered as a common sub-arrangement and the algorithm is now examining the pair of intervals with label “D”, then by applying the \otimes operation on those two parts, the resulting common pattern would be $\{C, D\}$, because intervals labeled “B” and “D” do not share the same relation in the two e-sequences. So, in our notation this translates to: $\{(2, 2), (3, 3)\} \otimes (4, 4) = \{(3, 3), (4, 4)\}$. This latter solution would not be stored in the DP array since it is contained within the maximal solution $\{(1, 1), (3, 3), (4, 4)\}$.

4.3.2 Properties of LCSP

Below we provide several key observations, insights and properties (their proofs are provided in Appendix) to demonstrate the correctness of Algorithm 1.

Clearly, solving the LCSP implies finding the CSP of maximum size. Furthermore, since LCSP is NP -complete, so unless $P = NP$, we should expect some part of the exact algorithm to perform exhaustive search. As explained above, for each $LCS(i, j)$ all $LCS(p, q)$ must be examined to discover the one that yields the actual longest common sub-pattern. In addition, for a single $LCS(i, j)$, multiple solutions may exist, e.g. for $LCS(3, 3)$, in the instance of Fig. 7, the two solutions are $\{A, C\}$ and $\{B, C\}$.

Algorithm 1 LCSP of Arrangements**Input:** Arrangements \mathcal{A}, \mathcal{B} **Output:** $\text{LCSP}(\mathcal{A}, \mathcal{B})$

```

for  $i = 1$  to  $|\mathcal{A}|$  do
  for  $j = 1$  to  $|\mathcal{B}|$  do
    if  $\text{label}(S_{A_i}) \neq \text{label}(S_{B_j})$  then
       $\text{LCS}(i, j) = \{\emptyset\}$ 
    else
      subarrangements =  $\emptyset$ 
      for all sub-problems  $(p, q)$  of  $(i, j)$  do
        for all solutions  $S_k(p, q)$  of  $\text{LCS}(p, q)$  do
          subarrangements = subarrangements  $\cup (S_k(p, q) \otimes (i, j))$ 
        end for
      end for
       $M = \text{keepMaximal}(\text{subarrangements})$ 
       $\text{LCS}(i, j) = M$ 
    end if
  end for
end for
return  $\arg \max_{i,j} \text{LCS}$ 

```

None of the solutions can be discarded since it is not clear which one would be the appropriate choice for the next sub-problems. Trying to match intervals with different labels does not result to valid solutions (for any $1 \leq i \leq m$, $1 \leq j \leq n$, if $E_{S_{A_i}} \neq E_{S_{B_j}}$ then $\text{LCS}(i, j) = \emptyset$), so we examine in detail the opposite case.

If $E_{S_{A_i}} = E_{S_{B_j}}$, we prove the correctness of the procedure based on the following properties.

Property 1 If all previous sub-problems $\text{LCS}(p, q)$ yield \emptyset as their solution, then $\text{LCS}(i, j)$ is composed only of one interval, that corresponds to S_{A_i} and S_{B_j} .

Property 2 The $\text{LCS}(p, q) \otimes (i, j)$ operation yields a common sub-pattern of $\{E_{S_{A_1}}, \dots, E_{S_{A_i}}\}$ and $\{E_{S_{B_1}}, \dots, E_{S_{B_j}}\}$.

Property 3 The $\text{LCS}(p, q) \otimes (i, j)$ operation yields a maximal CSP, that matches A_i to B_j , with the involved intervals up to that point.

The above properties guarantee that at a given step, maximal common sub-patterns are retrieved. However, we need to prove that the best solution is also retrieved.

Property 4 Given the set $\mathcal{M}_{i-1, j-1}$ of all maximal CSPs among $\{S_{A_1}, \dots, S_{A_{i-1}}\}$ and $\{S_{B_1}, \dots, S_{B_{j-1}}\}$, we can construct the set of maximal CSPs among $\{S_{A_1}, \dots, S_{A_i}\}$ and $\{S_{B_1}, \dots, S_{B_j}\}$ that match S_{A_i} to S_{B_j} , by keeping from each $\mu \in \mathcal{M}_{i-1, j-1}$ only the intervals whose corresponding intervals in \mathcal{A}, \mathcal{B} share the same relations with S_{A_i} and S_{B_j} respectively.

Conversely, in order to acquire $\text{LCS}(i, j)$, in the worst-case one needs to examine all previous maximal CSPs. However, if intervals S_{A_i} and S_{B_j} have a ‘follow’ relation with all previous intervals, then for $\text{LCS}(i, j)$, one only needs to consider the maximal

CSPs of maximum size over all pairs of sub-arrangements and append to them the new corresponding interval. So, we identify that the need to search exhaustively is restricted to the subproblems corresponding to pairs of intervals that have some overlap with S_{Ai} and S_{Bj} . Finally, since the method at point (i, j) returns the maximal CSP(s) whose last interval corresponds to S_{Ai} and S_{Bj} , the LCSP of the two whole arrangements is found by selecting the cell (i, j) which contains the largest sub-arrangement.

4.3.3 Extensions of Algorithm 1

We must note several key extensions of Algorithm 1. First, the algorithm can support constraints so that the retrieved LCSP does not pair together specific pairs of intervals. For example, while our problem formulation considers only relations among intervals, certain pairs of intervals may perhaps differ significantly in duration so that the user would not want to allow the algorithm to match them. This is achieved simply by extending accordingly the `if`-condition of Algorithm 1, and hence marking in the DP array the cells corresponding to the disallowed pairs with an empty set, as if the labels were different. For any disallowed pair, this will result in not producing any CSP that implies that they are matching counterparts.

The second extension is similar to the first but with the opposite intention. By replacing the function in the `if`-condition, that checks label equality, with a different function, we may allow common sub-arrangements assuming equivalence among intervals with different labels. This is more general than simply mapping them to new distinct labels.

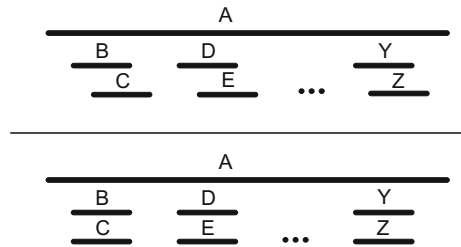
The third extension applies to the score returned by the algorithm. Since our DP algorithm computes many maximal common sub-patterns, and most importantly maintains the actual pairings, one may apply a custom cost or utility function on each found pattern and return the optimal under that function. For example, the cost function might be related to the difference in time duration of the paired intervals. This is a post-computation filtering step and for each DP array cell the algorithm will still compute maximal CSPs based on the interval relations. Hence, there are no guarantees that the retrieved LCSP is the optimal under all possible cost functions, nor do we study this particular problem any further in this work.

4.3.4 Complexity of Algorithm 1

For the DP algorithm, $LCS(i, j)$ must be computed for all possible pairs of i and j . To compute each $LCS(i, j)$, it is necessary to check all the solutions of all $O(|A| \cdot |B|)$ sub-problems; checking a solution requires linear time with respect to the size of the LCS, which is at most $\min\{|A|, |B|\}$. Hence, the total complexity of the algorithm is $O(n^3 \cdot m^2 \cdot s)$, if $n \leq m$, where s is the maximum number of solutions over all $LCS(i, j)$. This does not prove that we have found a polynomial time algorithm for NP -complete problems, since s , the number of solutions, can be exponential in the size of the input.

Computationally hard instances of Clique reduce to hard instances of LCSP. In such cases, the exact algorithm has to search among a number of solutions which is

Fig. 11 A category of arrangement pairs where any event-interval label appears exactly once, yet the number of candidate solutions stored at each point of the DP algorithm is exponential in the size of the arrangements



exponential in the size of the arrangements. Despite that, this is not a characteristic only of arrangements containing intervals which all have the same label.

In Fig. 11 we display a category of instances where any event-interval label appears exactly once, yet the number of candidate solutions stored at each point of the dynamic programming algorithm is exponential in the size of the arrangements. In particular, the number of partial solutions is $O(2^{\frac{n}{2}})$. There exist multiple LCSPs with size equal to half the size of the original arrangements. They contain only one event-interval for every pair of overlapping event-intervals in the first arrangement.

If one of the two arrangements, in an instance of LCSP, is a proper subset of the other, then using our exact algorithm one could extract all the appearances of the smaller arrangement in the larger. From an alternative scope, this implies that the performance of our exact algorithm can be improved by implementing run-time optimizations such as breaking once the smaller arrangement is found in the larger.

5 Polynomial time and space approximations

In Sect. 4.1 we proved that the LCSP problem belongs to the class of NP-hard problems. This makes the use of any exact algorithm impractical for handling all problem instances. Furthermore, we demonstrated that Clique is as hard as the decision version of LCSP. While it still remains open whether inapproximability results for Clique (Håstad 1996; Feige et al. 1991) are carried over to LCSP, the above reduction provides strong reasons to believe that they do. So, in our efforts to devise usable techniques that would yield approximate solutions, instead of focusing on designing fully polynomial-time approximation schemes (FPTAS), we restrict our aforementioned exact algorithm so that it terminates in polynomial time.

We transform the exact algorithm into a greedy algorithm. For each cell of the DP array, the algorithm maintains only the solutions of maximum size. For example, while computing $LCS(i, j)$, assume that the algorithm had discovered several solutions of size 4, and then discovers one of size 5. The latter is retained and all the other are purged. Similarly, while computing the values for the same cell, if a larger solution is found, those of size 5 are discarded. This prevents the algorithm from performing exhaustive search. However, Fig. 11 depicts an instance for which even this greedy approach will produce an exponential number of solutions w.r.t. the size of the input. Hence, we further force the algorithm to restrict the number of solutions it maintains at each cell of the DP array. In other words, we establish that the time complexity

$O(m^2 \cdot n^3 \cdot s)$ remains polynomial by ensuring that the value of s remains polynomial. At any step (i, j) of the algorithm, it is impossible to know in advance which solutions are the ones that would allow the greedy algorithm to reach the optimal solution.

We experiment with three natural strategies for limiting the number of solutions, namely *First_seen*, *Last_seen*, and *Random*. This restriction can be seen as having a buffer of limited size that is responsible for maintain the solutions.

More precisely, we have:

- *First_seen* At each step $LCS(i, j)$ and for a buffer of size b , the first b solutions of (only) maximum size are retained while the rest are ignored.
- *Last_seen* At each step $LCS(i, j)$ only the last b solutions of maximum size are retained.
- *Random* We randomly retain b solutions of maximum size. We achieve that by applying *Reservoir sampling* (Vitter 1985).

6 Upper bounds

Regardless of the efficiency of the approximations strategies presented earlier, their $\Omega(m^2 \cdot n^2)$ complexity is prohibitive for brute-force 1-NN searches in large-scale systems. In order to inexpensively prune many of the required comparisons, one of the common practices is to use upper-bounds (or lower-bounds for distance functions) (Vlachos et al. 2006); another common practice is of course indexing. In this section we define two upper bounds for LCSP.

The first upper bound requires linear time and relies on the count of common interval labels. Given an arrangement \mathcal{A} over the label alphabet Σ , we construct $u^{\mathcal{A}}$, a $|\Sigma|$ -dimensional vector that stores for each event label in Σ the count of event-intervals in \mathcal{A} that share that label.

Definition 5 (Upper bound UB_{CI}) Given arrangements \mathcal{A}, \mathcal{B} , the upper bound UB_{CI} is defined as

$$UB_{CI}(\mathcal{A}, \mathcal{B}) = \sum_{i=1}^{|\Sigma|} \min\{u_i^{\mathcal{A}}, u_i^{\mathcal{B}}\}$$

Proof Since the LCSP of \mathcal{A}, \mathcal{B} can be only a subset of the common intervals, the following holds:

$$UB_{CI}(\mathcal{A}, \mathcal{B}) \geq LCSP(\mathcal{A}, \mathcal{B}), \forall \mathcal{A}, \mathcal{B}.$$

□

UB_{CI} 's linear time complexity is the lowest possible. However, it focuses only on the labels and ignores any temporal order information. We can expect UB_{CI} to fail in cases where the dataset's arrangements contain a fixed or similar amount of intervals of every label. If a higher complexity is allowed, one can take advantage of the order of the intervals in each arrangement. This can be achieved by applying the LCS algorithm

for symbolic sequences. In this case each symbols would be the label of an intervals. However, for $O(nm)$ time complexity, a slightly tighter bound can be achieved, again using LCS but this time using the starting and ending point of the intervals instead of their labels. This approach imposes a constant factor of 4 to the complexity, since the symbols are twice as many as the intervals.

Given an e-sequence or arrangement \mathcal{A} , we can map it to its *semi-interval sequence* representation $\mathcal{C}^{\mathcal{A}}$ (Mörchen and Fradkin 2010) (a semi-interval is a tuple (E, t) , where E is the label interval and t is a time point). Every interval is replaced by two symbols corresponding to its start and end point. Such symbols can be the interval label followed by distinctive letters, for example ‘_s’ and ‘_e’, to distinguish between the start and end points. In our case, the order of the symbols is the order in which they happen. In case of concurrency, the symbols follow the partial order of the corresponding intervals. The absolute time values are discarded. An example of the semi-interval sequence is depicted in Fig. 4. We need to point out that this upper bound, namely UB_{CPS} , is equivalent to applying LCSS to our problem.

We formally define UB_{CPS} and prove that it is an actual upper bound by first proving the following lemma.

Lemma 2 *The semi-interval sequence representation of the LCSP of two arrangements \mathcal{A}, \mathcal{B} is a subsequence of the semi-interval sequences of both arrangements.*

Proof The intervals that belong to the LCSP appear in the same order in \mathcal{A}, \mathcal{B} , and as a result so do their start and end points. Consequently, the start- and end-symbols appear in the same order in the semi-interval sequence and they form a subsequence of length $2 * |LCSP(\mathcal{A}, \mathcal{B})|$. \square

Definition 6 (Upper bound UB_{CPS}) Given arrangements \mathcal{A}, \mathcal{B} , the upper bound UB_{CPS} of $LCSP$ is defined as

$$UB_{CPS}(\mathcal{A}, \mathcal{B}) = \frac{LCS(\mathcal{C}^{\mathcal{A}}, \mathcal{C}^{\mathcal{B}})}{2}.$$

Proof Suppose that UB_{LCSP} is not an upper bound to $LCSP$. Then the following does not hold:

$$UB_{CPS}(\mathcal{A}, \mathcal{B}) \geq LCSP(\mathcal{A}, \mathcal{B}), \forall \mathcal{A}, \mathcal{B}.$$

Then, there would exist a pair of arrangements such that the LCS of their semi-interval sequences is less or equal to twice the size of their LCSP. In such case, and by using Lemma 2, the semi-interval sequence corresponding to LCSP would be greater or equal to the LCS of the semi-interval sequences. This cannot hold. Hence, UB_{CPS} is a valid upper bound to LCSP. The above requires that concurrent semi-intervals are sorted in a consistent manner. \square

7 Experiments

We explored the effectiveness of the proposed polynomial approximation techniques, and demonstrated the efficiency of the proposed bounds by studying their pruning

Table 2 Statistical details of the used datasets

Dataset	No. of e-seq.	e-sequence size			No. of labels	No. of classes	Max e-seq. length	Interval size			
		Min.	Max.	Average				Mean	STD	Min	Max
ASL-BU	873	3	40	17	216	9	5901	594	590	3	4468
Auslan2	200	9	20	12	12	10	30	20	12	1	30
Blocks	210	3	12	6	8	8	123	17	12	1	57
Context	240	47	149	81	54	5	284	69	81	1	284
Hepatitis	498	15	592	108	63	2	7555	634	1093	1	7555
Pioneer	160	36	89	56	92	3	80	36	21	1	80
Skating	530	27	143	44	41	6	6829	576	672	1	6829
BigSynth	5000	20	20	20	5	0	149	6	6	1	44
Cliques	13	3	105	43	1	0	14	4	3	1	14

power and tightness. In addition, we conducted experiments to evaluate the performance of LCSP in terms of 1-NN and 3-NN classification and clustering against the standard state-of-the-art LCSS technique.

The datasets and source code used in our experiments are available online.¹

7.1 Experimental setup

For our experiments we used real and synthetic datasets.

7.1.1 Real datasets

For our experiments we used seven real datasets. A summary of the statistical details for each dataset is shown in Table 2. Below, we describe each dataset in more detail:

- ASL-BU (Papapetrou et al. 2009). Event labels correspond to grammatical or syntactic forms (e.g., wh-word, wh- question, verb, noun, etc.) as well as facial or gestural expressions (e.g., head tilt right, rapid head shake, eyebrow raise, etc.). An e-sequence is an expression of a sentence using sign language.
- Auslan2 (Mörchen and Fradkin 2010). The e-sequences were derived from the Australian Sign Language dataset available in the UCI repository.² Each event interval represents a word like girl or right.
- Blocks (Mörchen and Fradkin 2010). Event labels correspond to visual primitives obtained from videos of a human hand stacking colored blocks and describe which blocks are touched as well as the actions of the hand (e.g., contacts blue or red, attached hand red, etc.). Each e-sequence represents one of eight different scenarios including atomic actions, such as pickup, or complete scenarios, such as assemble.

¹ <http://users.ics.aalto.fi/kostakis/software/lcsp/>.

² <http://www.ics.uci.edu/mllearn/MLRepository.html>.

- Context (Mörchen and Fradkin 2010). Event labels were derived from categoric and numeric data describing the context of a mobile device carried by humans in different situations. Each e-sequence represents one of five different scenarios such as street or meeting.
- Hepatitis (Patel et al. 2008). The dataset contains information about patients who have either Hepatitis B or Hepatitis C. The event intervals represent the results of 63 regular tests. Each e-sequence describes a series of tests taken by a patient.
- Pioneer (Mörchen and Fradkin 2010). This dataset was constructed from the Pioneer-1 dataset available in the UCI repository. Event intervals correspond to the input provided by the robot sensors. Each e-sequence in the dataset describes one of three scenarios: gripping, move, turn.
- Skating (Mörchen and Fradkin 2010). Event intervals describe muscle activity and leg position of 6 professional In-Line Speed Skaters during controlled tests at 7 different speeds on a treadmill. Each e-sequence represents a complete movement cycle.

7.1.2 Synthetic datasets

We implemented a random e-sequence generator that takes as input certain parameters and produces a synthetic dataset of event-interval sequences. The generator, that takes as input several parameters, works as follows: an initial interval is produced. For each additional interval, one of the 7 relations is chosen uniformly and that is enforced between the new and the last interval. If the chosen relation does not completely define the boundaries of the interval (e.g. it is not a *matches* relation) then one or both are chosen randomly based on the input parameters. This allows to create cases of overlapping starting and endpoints and at the same time enables to define the expected lengths of the intervals. The label of each interval is also chosen uniformly from the given set. The source code is publicly available.³ By using our generator, we create a big synthetic dataset (BigSynth) intended mostly for scalability experiments.

We also manually create a smaller dataset, ‘Cliques’, and as the name suggests it corresponds to the e-sequence representation of cliques. In particular it contains all cliques of size 3 to 15. This is intended to test the tightness of our approximation algorithms. The statistics of both datasets are depicted in Table 2.

7.1.3 Evaluation metrics

We benchmark the three approximation policies for three different buffer sizes in terms of *approximation tightness*. We define *approximation tightness* as the value of the following ratio:

$$T_{approx} = \frac{Approx_{LCSP}(\mathcal{A}, \mathcal{B})}{LCSP(\mathcal{A}, \mathcal{B})}. \quad (3)$$

³ http://users.ics.aalto.fi/kostakis/software/intgen_lcsp.zip.

Table 3 Running time of exact LCSP algorithm

Dataset	Time
Asl	14''
Auslan2	1.5''
Blocks	<1''
Pioneer	15''
Context*	–(5 h 42')
Hepatitis*	–(8 h 25')
Skating*	–(15')
BigSynth	140h
Cliques	4 h 58'

* The datasets that contain hard instances and the values correspond to the *Random* approximation policy with $n/2$ buffer size

Since these policies are under-approximating LCSP, $T \in [0, 1]$. Clearly, we desire values closer to 1. To avoid favoring instances of smaller arrangements by having fixed sized limit on stored solutions, for each instance the buffer size was set to be equal to a fraction of the size of the shortest arrangement. The values we selected were $n/2$, $n/5$ and $n/10$.

We benchmark the two upper bounds in terms of tightness and pruning power under 1-NN search. The pruning power is the ratio of pruned comparisons under the LCSP algorithm for linear-scan 1-NN searches, when we first use the upper bound. Similarly to before, we define the tightness of an upper bound as the average value of the following fraction:

$$T_{UB} = \frac{UB(\mathcal{A}, \mathcal{B})}{LCSP(\mathcal{A}, \mathcal{B})}, \quad (4)$$

over all pairs of arrangements in the dataset. For the upper bounds, the tightness value's codomain is $[1, \infty)$. To distinguish among the two tightness values, if it is not clear from the context, we will refer to the first as *approximation tightness*.

7.2 Experimental results

When computing the similarity matrices of all datasets using the LCSP exact algorithm, we were able to complete the process for only 4 out of 7 datasets: Auslan2, Blocks, Pioneer and ASL. The rest of the datasets (Hepatitis, Context and Skating) contain “very hard instances”. With very hard instances we refer to those instances for which the algorithm requires more than 8GB of main memory (Java implementation). So, in our experiments we use those values given by the most expensive variations of all three polynomial approximation policies; when the number of stored solutions at each step are $n/2$.

The time required to compute the LCSP of all N^2 pairs of e-sequences, for each dataset, is depicted in Table 3. For the case of the datasets containing hard instances, the values in brackets denote the run-time of the **Random** approximation policy with $n/2$ buffer size.

Table 4 For each of the seven datasets we show the average approximation tightness for each of the three approximations

Dataset	First seen			Last seen			Random		
	n/2	n/5	n/10	n/2	n/5	n/10	n/2	n/5	n/10
ASL-BU	0.9999	0.9996	0.9967	0.9997	0.9994	0.9957	0.9999	0.9996	0.9967
Auslan2	1	0.9999	0.9997	1	0.9998	0.9997	1	0.9999	0.9997
Blocks	1	0.9995	0.9995	0.9999	0.9992	0.992	1	0.9995	0.9994
Context*	–	0.9997	0.9992	–	0.9996	0.9989	–	0.9998	0.9993
Hepatitis*	–	0.9985	0.9961	–	0.9979	0.9947	–	0.9991	0.9966
Pioneer	1	0.9999	0.9999	0.9999	0.9999	0.9998	1	1	0.9999
Skating*	–	0.9991	0.9963	–	0.9983	0.9926	–	0.9993	0.9961

* The datasets that contain hard instances and the values correspond to the *Random* approximation policy with $n/2$ buffer size

Table 5 For each of the seven datasets we show the minimum approximation tightness for each of the three approximations

Dataset	First seen			Last seen			Random		
	n/2	n/5	n/10	n/2	n/5	n/10	n/2	n/5	n/10
ASL-BU	0.75	0.667	0.5	0.75	0.667	0.5	0.75	0.667	0.5
Auslan2	1	0.91	0.8889	1	0.91	0.875	1	0.91	0.8889
Blocks	1	0.667	0.667	0.8	0.667	0.667	1	0.667	0.667
Context*	–	0.9063	0.9032	–	0.9117	0.909	–	0.9393	0.903
Hepatitis*	–	0.6667	0.6667	–	0.6667	0.6	–	0.6667	0.6667
Pioneer	1	0.9523	0.9167	0.9583	0.9166	0.9167	1	1	0.9167
Skating*	–	0.7272	0.6667	–	0.75	0.6	–	0.7777	0.69

* The datasets that contain hard instances and the values correspond to the *Random* approximation policy with $n/2$ buffer size

7.2.1 Polynomial approximation policies

The average approximation tightness for all datasets and all policies was between 99 and 100 %. In Tables 4 and 5 we show the average and minimum approximation tightness, respectively, for each of the seven datasets and three strategies studied in this paper.

As expected, a smaller buffer size resulted to worse values on average. For the harder instances we noticed that the *Random* policy provided slightly better results than *First_seen* and *Last_seen*. A reason for that is that the latter policies can be seen as hill-climbing heuristics that always make the same choice at every step of the solution space traversal. As a result, they restrict themselves to a specific region of the whole solution space and are bound to be trapped in local maxima. On the other hand, the *Random* policy does not restrict itself to a specific region of the solution space since it randomly selects solutions if needed. However, it faces the hazard of missing the global maximum if it randomly discards all the paths leading to it.

Table 6 Tightness and pruning power for UB_{CPS}

Dataset	1-NN pruning power	Tightness
ASL	99.2	1.26
Auslan2	89.8	1.15
Blocks	94.3	1.25
Pioneer	59.5	1.52
Context*	40.4	1.40
Hepatitis*	30.1	2.59
Skating*	55	1.64
BigSynth	52.4	1.54
Cliques	46.7	1.00

* The datasets that contain hard instances and the values correspond to the *Random* approximation policy with $n/2$ buffer size

Surprisingly, we also notice that in the datasets with very hard instances, there are few instances where a smaller buffer provided a better solution even for *First_seen* and *Last_seen*; for *Random* this is something to be expected due to the randomness in selecting solutions from the sub-problems. This is counter-intuitive but can be explained. By allowing more solutions to be stored at each (i, j) th step of the dynamic program, we allow those solutions to be the reason more solutions are created in succeeding steps. However, the latter solutions do not necessarily lead to the optimal solution. Instead, they act as noise and they compete for the buffer with those solutions that would allow to achieve the global optimum.

In terms of execution time, the speedup provided by the approximation policies for computing the distance matrix of the big synthetic dataset was 35.35, 50.35 and 67.3 times for buffers of size $n/2$, $n/5$ and $n/10$ respectively; *First_seen* was slightly faster than *Last_seen* and **Random**. Still, since our approximation algorithms require polynomial time and the exact algorithm is exponential in the worst case, as the size of the instances increases, the possible difference in running time is in the general case unbounded.

From the Cliques dataset, we witness that the approximation algorithms are able to achieve an accuracy of 100 %, for arrangements corresponding to cliques of up to size 15. However, when attempting to investigate their performance on larger cliques, we notice a computational blow up for cliques of size around 18. This is due to the fact that for such hard instances, the $O(n^6)$ complexity becomes a significant drawback.

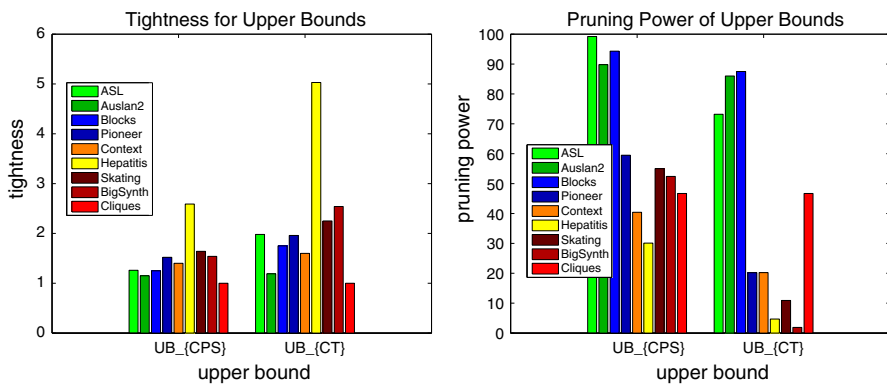
7.2.2 Upper bounds

The results of the tightness and pruning power experiments can be seen in Tables 6 and 7, as well as in Fig. 12. As mentioned previously, we were not able to compute the similarity matrices for the three datasets marked with an asterisk, so for those the values correspond to those obtained by the approximation policies. For UB_{CPS} we witnessed average tightness values between 1.15 and 2.59, while the average pruning power was from 30.1 to 99.2 %. For UB_{CI} , the values were worse, as expected. For the tightness, the average values observed were between 1.19 and 5.03. For the 3 datasets that we do not have the exact scores, the values might be much higher than the ground

Table 7 Tightness and pruning power for UB_{CI}

Dataset	1-NN pruning power	Tightness
Asl	73.2	1.98
Auslan2	86.0	1.19
Blocks	87.5	1.75
Pioneer	20.2	1.96
Context*	20.21	1.6
Hepatitis*	4.7	5.03
Skating*	10.9	2.25
BigSynth	1.9	2.54
Cliques	46.7	1.00

* The datasets that contain hard instances and the values correspond to the *Random* approximation policy with $n/2$ buffer size

**Fig. 12** Tightness (*left*) and pruning power (*right*) of the two upper bounds on all datasets

truth, since it is not clear by how much the policies are under-approximating LCSP. The average pruning power observed was between 1.9 and 87.5 %.

7.2.3 Scalability

We investigate the scalability of our methods on the big synthetic dataset. We monitor the time required for 1-NN queries, when using linear (brute-force) scan and the exact algorithm for LCSP, and we compare that to a linear scan with the use of UB_{CPS} for pruning. We don't consider the UB_{CI} since its pruning power is insignificant (see Table 7). The dataset does not contain any instances that our method cannot compute, so we do not employ any of the approximation strategies.

We witness that the time needed for computing each instance of UB_{CPS} is constant, given that all e-sequences contain the same number of intervals and as a result the same number of symbols when mapped to strings. On the other hand, there is a variable number of sub-solutions that need to be investigated when computing the LCSP instances.

The total time required to perform all 1-NN queries for the synthetic dataset via a linear scan requires 95.5 % more time than if UB_{CPS} is used first.

Table 8 Classification accuracy of LCSS and LCSP under 1-NN and 3-NN classification schemes

	LCSS 1NN (%)	LCSP 1NN (%)	LCSS 3NN (%)	LCSP 3NN (%)
ASL	73.23	63.57	68.5	58.17
Auslan2	26	27	24	29.0
Blocks	81.9	80.95	91.42	87.61
Context*	77.91	86.6–87.08	73.75	85.83–86.25
Hepatitis*	66.26	66.66–68.07	68.47	70.88–71.28
Pioneer	95	97.5	93.75	96.25
Skating*	76.22	93.02	71.69	92.08–93.02

Bold values indicate the best performance for each experiment

* The datasets that contain hard instances and the values correspond to the *Random* approximation policy with $n/2$ buffer size

7.3 Comparison to LCSS: k-NN classification and clustering

LCSS solves a problem similar to LCSP, that is easier to compute. Hence, we need to justify the need for LCSP. The fundamental difference between the two techniques is that they are defined for different types of sequences, and hence different types of longest common subpatterns. For the case of LCSP the subpattern is an arrangement, while for LCSS the subpattern is a symbolic sequence. In the previous sections we examined the performance of LCSS (which is equivalent to UB_{CPS} , as pointed out in Sect. 6), and showed that it can be used as a pruning technique for computing the correct LCSP. Hence it becomes apparent that LCSS does not and cannot compute LCSP.

For providing a more extensive comparison between LCSS and LCSP, we further proceed to benchmark the two techniques for the tasks of k-NN classification and clustering. The results clearly demonstrate that for most cases LCSS is inferior to LCSP, and hence LCSP should be preferred.

First, LCSS was used for 1-NN and 3-NN classification. For each dataset, we consider each e-sequence as a query and the remaining e-sequences as the database. The class of query is assigned to be the class of its nearest neighbor, i.e., the database e-sequence with the highest similarity score. Hence, the classification accuracy is then the fraction of e-sequences in the dataset that are correctly classified using the remaining ones. The results are depicted in Table 8. We see that LCSS outperforms LCSP only for the cases of “ASL” and “Blocks” dataset and this happens for both 1-NN and 3-NN. For all remaining cases, LCSP provides better classification accuracy for both 1-NN and 3-NN respectively.

Next, we evaluated the performance of LCSS and LCSP for the task of clustering. Since we do not have a feature space, we employed *k-medoids*. The number of clusters (the value of k) was set equal to the predefined number of classes for each dataset (see Table 2). For each dataset, we performed 10 clustering sessions, and for each session the basic algorithm was executed 1000 times, while the best solution was retained. Since both LCSP and LCSS provide similarity scores, those were transformed to distances

Table 9 Average clustering purity of LCSS and LCSP under k-medoids

	LCSS	LCSP
Auslan2	0.17	0.25
Blocks	0.61	0.61
Context*	0.46	0.54
Hepatitis*	0.61	0.59
Pioneer	0.64	0.73
Skating*	0.58	0.69

Bold values indicate the best performance for each experiment

* The datasets that contain hard instances and the values correspond to the *Random* approximation policy with $n/2$ buffer size

by computing $1 - \frac{s(S_1, S_2)}{\min(|S_1|, |S_2|)}$, where $s(S_1, S_2)$ is the value of LCSS or LCSS between sequences S_1 and S_2 . For the clustering evaluation metric, we compute the clustering *purity*. The average clustering purity for each dataset over the 10 sessions is depicted in Table 9; for ASL we omit the results since each sample may belong to multiple classes. We observe that LCSS yields better values only for the Hepatitis dataset, while for the Blocks dataset the values are similar. For the rest of the datasets, LCSP clearly outperforms LCSS.

8 Use-cases of LCSP

In this section we describe several use-cases to demonstrate the need and suitability of LCSP. Furthermore, we demonstrate how LCSP is a more appropriate distance measure than full-sequence matching. We perform some pilot studies related to classification via profiling, system verification and anomaly detection for the fields of sign language (ASL) and sensor data (Pioneer robot).

8.1 Sign language profile-based classification

We demonstrate the suitability and applicability of LCSP for performing profile searching and classification in the sign language domain based on given sign language profiles.

In a sign language setting the objective is to classify a set of unknown e-sequences based on the existence of a “profile” arrangement that determines the class label of each e-sequence. Examples of profile arrangements are given in Fig. 13, left part, where the profiles are characteristic arrangements that describe American Sign Language e-sequences of the class “Wh-question” (Papapetrou et al. 2009). Note that such profiles are, in general, expected to be much shorter than the target unclassified e-sequences. For example, in our ASL dataset, the average e-sequence size is 17 while the maximum is 40, whereas the size of a typical ASL profile is usually not longer than 4 (Papapetrou et al. 2009). Given a set of profiles, for each e-sequence in the dataset we examine whether it contains one or more of these profiles. If so, then the e-sequence is assigned with the corresponding class(es). Determining the existence of

Wh-question	Combined
<div> <div>Wh-word</div> <div>Head pos: forward</div> <div>Wh-question</div> </div>	<div> <div>Wh-word</div> <div>Head pos: forward</div> <div>Lowered eye-brows</div> <div>Wh-question</div> </div>
<div> <div>Wh-word</div> <div>Lowered eye-brows</div> <div>Wh-question</div> </div>	

Fig. 13 Example of two profile arrangements for a “Wh-Question” (*left*) in American Sign Language, and their combination

a profile in an e-sequence is accomplished by LCSP; simply by checking if their LCSP is as long as the profile. Clearly, this check is impossible to perform via full-sequence matching algorithms, such as Artemis (Kostakis et al. 2011), since their objective is orthogonal to that of LCSP. Furthermore, we witnessed in all cases that, under Artemis, the NN ranks of the profiles are arbitrary, since all points in the query are “forced” to match a database counterpart, while differences in size between query and database sequences also highly distort the matching. An additional advantage of using LCSP for this task is that several profiles can be combined into a single profile, and then a threshold check is sufficient. For example, the profiles in Fig. 13 can be combined into one, making it sufficient to check whether the LCSP of the combined profile and an unknown e-sequence contains 3 or more intervals. LCSP achieved 100 % precision, which was expected. The reason for that is simply that if the profile arrangements exist in a classified arrangement, then LCSP is able to fully detect them and make the correct assignment(s). By adding more profile arrangements to cover all cases, the recall of the classification increases and all the relevant e-sequences may be retrieved.

This methodology may be applied to a wide range of applications by selecting the appropriate profile sequences.

8.2 System verification

Next, we demonstrate the applicability of LCSP for system verification. Run-time system verification can be seen as a special case of classification. E-sequences would denote observed telemetry or executions of a system or program. In the task of system verification, each observed e-sequence would fall into either the category of “acceptable” or “unacceptable” executions. Suppose we have defined a set of “undesired properties” in the form of arrangements. We need to determine whether they are present in the e-sequence that is being examined. If one or more such properties are contained in whole, or above certain threshold, then the e-sequence is regarded as a violating execution and the system does not comply to the required specifications. Examples of such applications include robot sensors, and sensor networks in general, as well as the execution of computer programs.

For the case of the Pioneer dataset, suppose that an undesirable property is that the robot’s gripping mechanism moves upwards. The naive approach to detect this would

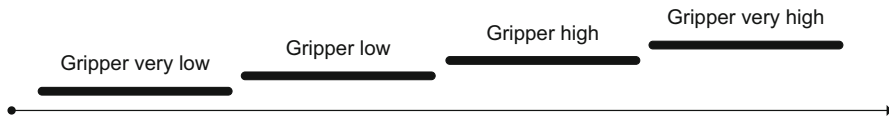


Fig. 14 Arrangement corresponding to the “undesired property” of the Pioneer’s gripping mechanism moving upwards. Each interval corresponds to a different state of the gripper

be to create a list of ‘if’ clauses for all $\binom{k}{2}$ possible pairs of states and then to check the values of all $\binom{t}{2}$ possible pairs of time points (k is the number of different states, t the duration of the e-sequence in time units). Instead, we can use LCSP and the property arrangement of the gripping mechanism moving upwards, as depicted in Fig. 14. Simply, if the LCSP between the robot’s execution monitoring e-sequence and the property has length greater than 1, regardless of the specific intervals, then one can be sure that the mechanism has moved upwards. Consequently, the execution should be classified as non-conforming to the requirements. In the Pioneer dataset, we are able to retrieve 6 e-sequences that contain an upward movement. We can apply the same approach for the gripper’s downward movement. This yields 10 e-sequences in which the gripper moves downwards. The two retrieved sets of e-sequences, that happen to be disjoint, constitute the whole set of e-sequences that belong to the ‘gripper’ class of the dataset. This approach would not have been possible when using full-sequence matching.

Similarly, if we were to verify the correctness of a controlled execution environment (sandbox) for computer programs, where intervals correspond to the time that a particular function is active (in the stack frame), the Fig. 4b would correspond to an undesirable property. This is due to the fact that a called function cannot return before the return of any function that has been called from within itself. Hence, if the LCSP of the execution e-sequence with that property has length greater or equal to 2, it would prove that our system malfunctions.

8.3 Anomaly detection

LCSP can also be applied for the purpose of anomaly detection. Suppose that we have a set of “necessary behaviours” in the form of arrangements. Using LCSP, we are able to potentially detect the absence, in whole or fraction, of a series of such arrangements. In the case of robot sensor data (Pioneer dataset), we could use these predefined arrangements to detect anomalies in robot movement data that correspond to, e.g., the robot moving straight and not turning. A requirement for the robot to move straight is having both its wheels rotate with the same velocity simultaneously. Hence, in such dataset an event label would represent both the wheel location and its velocity range, e.g., *RW-Vel-low* would correspond to the Right Wheel moving with low velocity, while *LW-Vel-high* would correspond to the Left Wheel moving with high velocity. It also becomes apparent that these events will have a time duration and hence each wheel velocity will correspond to an event interval, e.g., the left wheel is moving with low velocity for 10 seconds. Based on the above, our requirement that the robot is moving straight is fulfilled when the event intervals corresponding to different wheel velocities “match”.

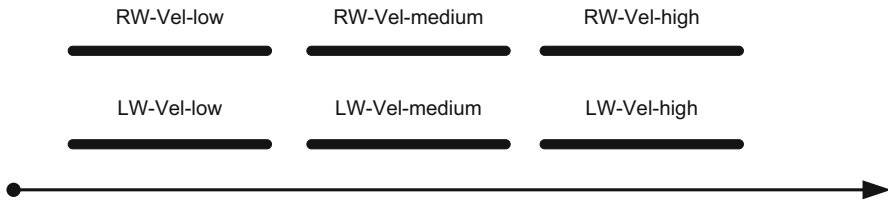


Fig. 15 Behaviour arrangement for confirming that the Pioneer robot moves straight. Each interval of the *top row* “matches” its counterpart in the *bottom row*. The intervals denote the angular velocity of the right and left wheels

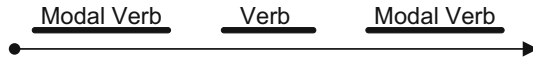


Fig. 16 Combined behaviour arrangement for confirming that modal verbs are used correctly in ASL

Using the “necessary behaviour” depicted in Fig. 15 as an arrangement of six events, we expect that the LCSP of this behaviour arrangement, call it \mathcal{Q} , and any e-sequence \mathcal{S} corresponding to moving straight should have an even amount of intervals ($|\text{LCSP}(\mathcal{S}, \mathcal{Q})| = 2k, k \in \mathbb{Z}^+$); one or more pairs of intervals and for each pair, each interval corresponding to one of the wheels. From our Pioneer dataset, we discover 14 out of 102 e-sequences of the specific class are being detected as anomalous. Manual verification of the result revealed that the intervals in question exhibited an ‘overlap’ relation instead of ‘match’, offset by a few time points (this is possibly due to noise in the recording phase, or when transforming the original data to e-sequences). Clearly, such approach cannot be applied when using a full-sequence matching algorithm.

Similarly, we can implement an ASL “grammar checker”. By maintaining a list of grammatical rules in the form of arrangements, we then use LCSP to determine the conformance of the user input to those rules. Suppose we would like to verify the grammar rule for modal verbs. In ASL grammar, modal verbs come before or after the main verb of the clause. To verify the correctness of the signed utterance, we can create two rules, one for each case. Then by using LCSP we should verify that either one appears as whole in the utterance, otherwise the grammar is incorrect. Alternatively, we can combine the two cases into a single rule, as depicted in Fig. 16. In this case, the check must be that the size of the LCSP between the rule and the signed utterance must be of size at least 2. Applying the described approach on our ASL dataset, we discovered two utterances did not follow this rule; it happens that in both of these cases the word “rain” is denoted as a noun instead of a verb.

9 Conclusions

We formally defined the problem of finding the LCSP problem for sequences of event-intervals. In addition, we proved that the LCSP problem belongs to the complexity class of NP -hard problems by showing that Clique can be reduced to it under a log-space reduction. This was achieved by establishing that arrangements of temporal intervals can be used to encode graphs. We also proved that $\text{LCSP} \in NP$ by proving that it reduces to Max Clique. Moreover, we introduced an exact algorithm for solving the LCSP problem. Furthermore, we proposed three policies for under-approximating

hard instances of LCSP and constructed the two upper-bounds for speeding up 1-NN searches under LCSP. Finally, we experimented on seven real datasets taken from various domains, including sign language, medicine, human motion, and sensor networks, and two synthetic datasets.

There are several directions for future work. On the practical side, we would be interested in experimenting with the use of LCSP in more real-world applications and examine the benefits it provides in comparison to the use of symbolic sequences or time-series. On the theoretical part, we demonstrated that Clique reduces to the decision version of LCSP. So, for the near future we plan to examine whether the inapproximability results demonstrated in the past for Clique hold for LCSP, too. However, a stronger result that we demonstrated is the fact that arrangements of temporal intervals can in certain cases be viewed as a generalization of graphs. Given the strong interest recently in graph mining and event-interval sequence mining, the most significant question that arises is how many of the ideas, algorithms and theoretical results can be exchanged between those two fields.

Finally, modifying the problem we just studied, we would be interested in devising fast algorithms for exact sub-sequence matching. We are curious whether we will discover the same complexity bounds as the MCS problem.

Appendix: Proof of the properties described in Sect. 4.3.2

1. Supposing that $LCS(i, j)$ was composed of more than one interval, then there must exist a pair of intervals with the same label in $\{S_{A1}, \dots, S_{Ai-1}\}$ and $\{S_{B1}, \dots, S_{Bi-1}\}$. That is a contradiction since it would imply that not all previous sub-problems yield \emptyset as their solution.
2. By applying the operation $LCS(p, q) \otimes (i, j)$ or, equivalently selecting from $LCS(p, q)$ only the intervals that induce similar relations to the corresponding interval of i and j , we make sure that the interval corresponding to i and j has the same relations to the previous intervals in the produced arrangement. Conversely, the existing intervals have the same relations to the correspondent of i and j . Additionally, pairs of existing intervals of $LCS(p, q)$ have identical relations with their correspondents in \mathcal{A} and \mathcal{B} ; this was examined when each interval was added to the solution of the previous sub-problems.
3. In other words, the \otimes operator does not discard extra intervals. Suppose that the maximal CSPs are correctly retrieved for all previous sub-problems $LCS(p, q)$, but not for $LCS(i, j)$. This would imply that an interval belonging to a maximal CSP of $\{E_{SA1}, \dots, E_{SAi}\}$ and $\{E_{SB1}, \dots, E_{SBj}\}$ (where A_i is matched to B_j) exists but was not selected for $LCS(i, j)$. But since the not-selected interval belongs to a maximal CSP then it has the same relation to S_{Ai} and S_{Bj} . So, since the relations are the same, the interval would have been selected for $LCS(i, j)$, which contradicts to the previous. Thus, the algorithm at point (i, j) returns maximal CSPs of $\{E_{SA1}, \dots, E_{SAi}\}$ and $\{E_{SB1}, \dots, E_{SBj}\}$ that matches E_{SAi} to E_{SBj} .
4. Suppose there exists a maximal CSP that matches A_i to B_j but was not discovered. This would imply that by removing the interval corresponding to A_i and B_j , one is

left with common a sub-pattern s . Then, either $s \subseteq r$, $r \in \mathcal{M}_{i-1,j-1}$ or not. In the first case, s must have been retrieved when performing $r \otimes (i, j)$, so this cannot be. So, it can only be that s is maximal but then it must hold that $s \in \mathcal{M}_{i-1,j-1}$. Contradiction.

An alternative approach is that in the Cartesian graph G_{AB} (see proof of Theorem 2 for exact definition), this corresponds to finding all maximal cliques containing the vertex u labeled (i, j) by checking all previously found maximal cliques and for each one returning its intersection with the neighbors of u .

References

- Abraham T, Roddick JF (1999) Incremental meta-mining from large temporal data sets. In: ER '98: Proceedings of the workshops on data warehousing and data mining, pp 41–54
- Ale JM, Rossi GH (2000) An approach to discovering temporal association rules. In: Proceedings of the 15th ACM symposium on applied computing, pp 294–300
- Allen J, Ferguson G (1994) Actions and events in interval temporal logic. *J Log Comput* 4:531–579
- Allen JF (1983) Maintaining knowledge about temporal intervals. *Commun ACM* 26(11):832–843
- Berendt B (1996) Explaining preferred mental models in Allen inferences with a metrical model of imagery. In: Proceedings of the 18th annual conference of the cognitive science society, pp 489–494
- Bergen B, Chang N (2005) Embodied construction grammar in simulation-based language understanding. *Constr Gram* 3:147–190
- Chen X, Petrounias I (1999) Mining temporal features in association rules. In: Proceedings of the 3rd European conference on principles and practice of knowledge discovery in databases. Springer-Verlag, New York, pp 295–300
- Chen YC, Peng WC, Le SY (2011) CEMiner—an efficient algorithms for mining closed patterns from interval-based data. In: Proceedings of the IEEE international conference on data mining (ICDM)
- Cormen TH, Rivest RL, Leiserson CE, Stein C (2001) Introduction to algorithms. MIT Press, Cambridge
- Feige U, Goldwasser S, Lovasz L, Safra S, Szegedy M (1991) Approximating clique is almost NP-complete. In: Proceedings of the 32nd annual IEEE symposium on foundations of computer science, pp 2–12
- Fradkin D, Moerchen F (2010) Margin-closed frequent sequential pattern mining. In: Proceedings of the ACM SIGKDD workshop on useful patterns. ACM, New York, UP '10, pp 45–54. doi:[10.1145/1816112.1816119](https://doi.org/10.1145/1816112.1816119)
- Giannotti F, Nanni M, Pedreschi D (2006) Efficient mining of temporally annotated sequences. In: Proceedings of the 6th SIAM data mining conference, vol 124, pp 348–359
- Håstad J (1996) Clique is hard to approximate within $n^{1-\epsilon}$. In: FOCS, pp 627–636
- Höppner F (2001) Discovery of temporal patterns—learning rules about the qualitative behaviour of time series. In: Proceedings of the 5th European conference on principles of knowledge discovery in databases, pp 192–203
- Höppner F, Klawonn F (2001) Finding informative rules in interval sequences. In: Proceedings of the 4th international symposium on advances in intelligent data analysis, pp 123–132
- Hwang SY, Wei CP, Yang WS (2004) Discovery of temporal patterns from process instances. *Comput Ind* 53(3):345–364
- Jiang D, Pei J (2009) Mining frequent cross-graph quasi-cliques. *ACM Trans Knowl Discov Data* 2(4):16:1–16:42
- Kam P, Fu AW (2000) Discovering temporal patterns for interval-based events. In: Proceedings of the 2nd international conference on data warehousing and knowledge discovery, pp 317–326
- Kosara R, Miksch S (2001) Visualizing complex notions of time. *Stud Health Technol Inf* 1:211–215
- Kostakis O, Papapetrou P, Hollmén J (2011) Artemis: assessing the similarity of event-interval sequences. In: Proceedings of the conference on machine learning and knowledge discovery in databases (ECML/PKDD 2011), pp 229–244
- Kotsifakos A, Papapetrou P, Athitsos V (2013) IBSM: interval-based sequence matching. In: Proceedings of the SIAM conference on data mining (SDM), pp 596–604
- Lam HT, Mrchen F, Fradkin D, Calders T (2014) Mining compressing sequential patterns. *Stat Anal Data Min* 7(1):34–52. doi:[10.1002/sam.11192](https://doi.org/10.1002/sam.11192)

- Laxman S, Sastry P, Unnikrishnan K (2007) Discovering frequent generalized episodes when events persist for different durations. *IEEE Trans Knowl Data Eng* 19(9):1188–1201. doi:[10.1109/TKDE.2007.1055](https://doi.org/10.1109/TKDE.2007.1055)
- Lin JL (2003) Mining maximal frequent intervals. In: *Proceedings of the 18th ACM symposium on applied computing*, pp 624–629
- Liu G, Wong L (2008) Effective pruning techniques for mining quasi-cliques. In: *Proceedings of the European conference on machine learning and knowledge discovery in databases: part II*. Springer-Verlag, Berlin, ECML PKDD '08, pp 33–49. doi:[10.1007/978-3-540-87481-2_3](https://doi.org/10.1007/978-3-540-87481-2_3)
- Mooney C, Roddick JF (2004) Mining relationships between interacting episodes. In: *Proceedings of the 4th SIAM international conference on data mining*
- Mörchen F (2007) Unsupervised pattern mining from symbolic temporal data. *SIGKDD Explor Newsl* 9:41–55
- Mörchen F, Fradkin D (2010) Robust mining of time intervals with semi-interval partial order patterns. In: *Proceedings of the 10th SIAM international conference on data mining*, pp 315–326
- Pachet F, Ramalho G, Carrive J (1996) Representing temporal musical objects and reasoning in the MusES system. *J New Music Res* 25(3):252–275
- Papapetrou P, Kollios G, Sclaroff S, Gunopulos D (2009) Mining frequent arrangements of temporal intervals. *Knowl Inf Syst* 21:133–171
- Patel D, Hsu W, Lee M (2008) Mining relationships among interval-based events for classification. In: *Proceedings of the 28th ACM SIGMOD international conference on management of data*, ACM, pp 393–404
- Paterson M, Dancik V (1994) Longest common subsequences. In: *Proceedings of the 19th MFCS*, number 841 in LNCS, pp 127–142
- Pissinou N, Radev I, Makki K (2001) Spatio-temporal modeling in video and multimedia geographic information systems. *GeoInformatica* 5(4):375–409
- Smith TF, Waterman MS (1981) Identification of common molecular subsequences. *J Mol Biol* 147:195–197
- Tsourakakis CE, Bonchi F, Gionis A, Gullo F, Tsiarli MA (2013) Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: *Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining*, pp 104–112
- Villafane R, Hua KA, Tran D, Maulik B (2000) Knowledge discovery from series of interval events. *Intell Inf Syst* 15(1):71–89
- Vitter JS (1985) Random sampling with a reservoir. *ACM Trans Math Softw (TOMS)* 11(1):37–57
- Vlachos M, Hadjieleftheriou M, Gunopulos D, Keogh EJ (2006) Indexing multidimensional time-series. *VLDB J* 15(1):1–20
- Winarko E, Roddick JF (2007) Armada—an algorithm for discovering richer relative temporal association rules from interval-based data. *Data Knowl Eng* 63(1):76–90. doi:[10.1016/j.datak.2006.10.009](https://doi.org/10.1016/j.datak.2006.10.009)
- Wu SY, Chen YL (2007) Mining nonambiguous temporal patterns for interval-based events. *IEEE Trans Knowl Data Eng* 19(6):742–758. doi:[10.1109/TKDE.2007.190613](https://doi.org/10.1109/TKDE.2007.190613)