Z-Embedding: A Spectral Representation of Event Intervals for Efficient Clustering and Classification

Zed Lee[1,2] ✉, Šarūnas Girdzijauskas[2], and Panagiotis Papapetrou[1]

[1] DSV, Stockholm University, Stockholm, Sweden*
{zed.lee, panagiotis}@dsv.su.se
[2] EECS, KTH Royal Institute of Technology, Stockholm, Sweden
{zed, sarunasg}@kth.se

Abstract. Sequences of event intervals occur in several application domains, while their inherent complexity hinders scalable solutions to tasks such as clustering and classification. In this paper, we propose a novel spectral embedding representation of event interval sequences that relies on bipartite graphs. More concretely, each event interval sequence is represented by a bipartite graph by following three main steps: (1) creating a hash table that can quickly convert a collection of event interval sequences into a bipartite graph representation, (2) creating and regularizing a bi-adjacency matrix corresponding to the bipartite graph, (3) defining a spectral embedding mapping on the bi-adjacency matrix. In addition, we show that substantial improvements can be achieved with regard to classification performance through pruning parameters that capture the nature of the relations formed by the event intervals. We demonstrate through extensive experimental evaluation on five real-world datasets that our approach can obtain runtime speedups of up to two orders of magnitude compared to other state-of-the-art methods and similar or better clustering and classification performance.

Keywords: event intervals · bipartite graph · spectral embedding · clustering · classification.

1 Introduction

The problem of sequential pattern mining has been studied in many application areas, and the main goal is to extract frequent patterns from event sequences [10, 22] or cluster instances that are characterized by similar patterns [13]. However, a major limitation of traditional sequential pattern mining algorithms is their assumption that events occur instantaneously. As a result, they fail to capture temporal relations that may occur between events, in application areas where

* This work was partly supported by the VR-2016-03372 Swedish Research Council Starting Grant, as well as the EXTREME project funded by the Digital Futures framework.

events have a time duration. To overcome this limitation, a wide body of research has been developed where the notion of event is extended to that of event interval. The advantage of this representation is that it can model event durations and hence the relationships between different event intervals based on their relative time positions, providing further insights into the nature of the underlying events. A wide range of application areas employ event interval representations, including including medicine [18], geoinformatics [14], or sign language [11].

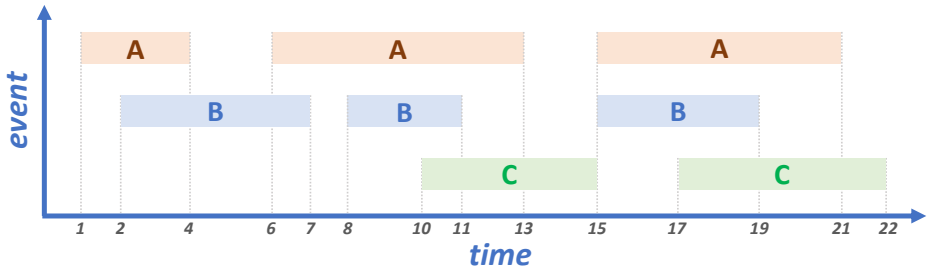

Fig. 1. Example of an sequence of eight event intervals of three event labels (A, B, and C). The time-span of the sequence is 22 time points.

Example. Fig. 1 represents a sequence of nine event intervals defined over an alphabet of three labels, A, B, and C. Each event interval is characterized by a specific label, as well as its start and end time points. Over time, the same event label can appear multiple times, and a different type of temporal relation can occur between each pair of event labels.

A multi-set of event intervals arranged in chronological order constitutes an *event interval sequence*. Relative relations between many events that occur within a sequence can lead to various forms of temporal compositions, e.g., by considering the temporal relations between the events using Allen's temporal logic [1]. In this setting, one challenging problem is finding robust and computationally cheap distance or similarity functions that can effectively capture the commonalities between pairs of such complex sequences. Such functions can then be used to provide scalable solutions to the problems of clustering and classification.

1.1 Related work

Early research on classification and clustering of sequences of temporal intervals has been focusing mostly on defining proper distance functions between the sequences. One such distance metric is *Artemis* [7], which calculates the distance between two event interval sequences by computing the ratio of temporal relations of specific pairs that both sequences have in common. In this case, the absolute time duration of each sequence is ignored, which makes *Artemis* oblivious to the length of the sequences. However, this method requires a substantial amount of computation time for checking all temporal relations occurring before and after

each and every event interval in a sequence. The application of this metric to k-NN classification demonstrated promising predictive performance.

An alternative measure, called *IBSM* [8], calculates the distance between two event interval sequences by constructing a binary matrix per sequence that is used to monitor the active event labels at each time point without explicitly considering any temporal relations between the events. Each time point is represented by a binary vector of size equal to the number of event labels in the alphabet, having its cells set to 1 for those labels that are active during that time point, while all other cells are set to 0. The main difference between *IBSM* and *Artemis* is that the former considers the time duration of each event interval to be crucial for the distance computation while the latter only considers their temporal relations irrespective of time duration. Moreover, in the case of *IBSM*, the distance computation between the binary matrices is fast. Nonetheless, since each sequence's absolute time points are different, extra processing time is required for interpolating the sequences to match their size. As a result, the time computation can be substantially slower when it involves sequence pairs with the highly disproportional number of event labels and time durations.

Recently, *STIFE* [2] has been proposed for classifying sequences of event intervals by using a combination of static and temporal features extracted from the sequences. The temporal features include pairs of event intervals that can achieve high class-separation (e.g., with respect to information gain). Nonetheless, the feature extraction time can make the algorithm even slower than *IBSM*. Furthermore, the extracted features are calibrated for feature-based classification and cannot be directly applied to other tasks, such as clustering.

Finally, the connection between graphs and temporal intervals is demonstrated by converting dynamic graphs to event intervals [6]. In this paper, we demonstrate that our proposed bipartite graph representation can be used to define a feature space at a substantially lower computational cost by using *graph spectral embeddings*, hence addressing the aforementioned scalability deficiencies of the three main competitors, i.e., *ARTEMIS*, *IBSM*, and *STIFE*. Graph spectral embeddings constitute a common clustering technique in graph mining for capturing community structure in graphs [12, 20, 21]. For bipartite graphs, bi-spectral clustering is introduced to speed up the embedding process using the bi-adjacency matrix, which removes the space for edges between instances in the same set from adjacency matrix [9], and its variants have been introduced on stochastic block model [24]. In recent years, the technique of regularizing this affinity matrix has been actively studied [3, 15] and shown to work well in terms of eigenvector perturbation on stochastic block model [5], conductance [23], and sensitivity to outliers [4]. This embedding space of an affinity matrix can also be used as a feature space for classification, showing better performance than previous distance metrics [17].

1.2 Contributions

In this paper, we propose a time-efficient approach for mapping event interval sequences to the spectral feature space of a bipartite graph representing the orig-

inal sequences. We additionally introduce space pruning techniques for achieving further speedups for both classification and clustering. The main contributions of this paper are summarized as follows:

- **Novelty.** We propose a novel three-step framework for representing sequences of event intervals by (1) constructing a search-efficient data structure (`G-HashTable`), (2) mapping the e-sequences to bipartite graphs, and (3) exploiting the bipartite graph representation to eventually map them to their corresponding spectral graph embedding space (`Z-Embedding`);
- **Efficiency.** The proposed three-step mapping results in a new feature space constructed with substantially less computation time (up to a factor of 200) compared to existing state-of-the-art representations;
- **Flexibility.** the proposed representation additionally exploits vertical and horizontal supports to represent the nature of the interval data space better, as well as three pruning parameters {`maxSup, minSup, gap`} for adding flexibility to the targeted features;
- **Clustering performance.** We demonstrate that the proposed representation can achieve higher clustering purity values than earlier methods for clustering event interval sequences on five real-world datasets;
- **Classification performance.** Using the same datasets, we demonstrate that the proposed feature space can achieve comparable classification performance against state-of-the-art classification methods for event interval sequences while maintaining substantially low computation time.

2 Background

Let $\Sigma = \{e_1, \ldots, e_m\}$ be a set of m event labels. An event that occurs during a specific time duration is called *event interval*. A set of event intervals for the same entity (e.g., a patient in a medical dataset) forms an *event-interval sequence* or *e-sequence*. Next, we provide more formal definitions for these two concepts.

Definition 1. (event interval) *An event interval $s = (e, t_s, t_e)$ is defined as a tuple of three elements, where $s.e \in \Sigma$ and $s.t_s$, $s.t_e$ define the start and end time of the interval, with $s.t_s \leq s.t_e$.*

In the special case where $s.t_s = s.t_e$, the event interval is *instantaneous*.

Definition 2. (e-sequence) *An e-sequence $\mathcal{S}$=$\{s_1, \ldots, s_n\}$ is a collection of event intervals. Event intervals in an e-sequence are sorted chronologically and can contain the same event label multiple times. More concretely, they first follow an ascending order based on their start time. If the start times are the same, the end times are arranged in ascending order. If the end times are also the same, they follow the order in which event labels are lexicographically sorted.*

We consider seven temporal relations (see Fig. 2), defined in the following set, $\mathcal{I} = \{$*follows*, *meets*, *overlaps*, *matches*, *contains*, *left-matches*, *right-matches*$\}$. Formally, a *temporal relation* $\mathcal{R}$ between two event intervals s_a and s_b, with

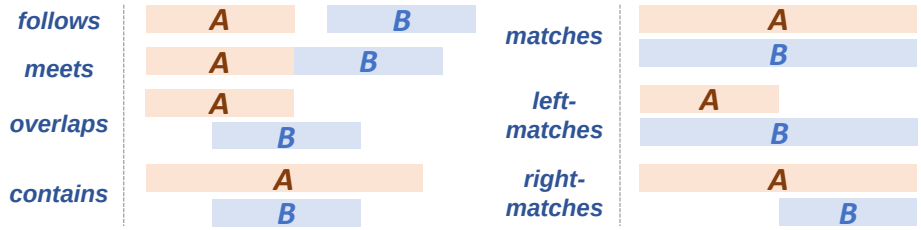

Fig. 2. Seven temporal relations that two event intervals can have, as defined in Allen's temporal logic [1].

$s_a.e = e_i, s_b.e = e_j$, is defined as a triplet $< e_i, e_j, r >$, with $r \in \mathcal{I}$. In [1], the number of relations is 13, including six inverse relations, but it can be reduced to seven by forcing the order by start time and removing six inverse relations except for *matches*, which does not have an inverse form. In several applications, we may not be interested in the absolute time values of event intervals but rather in the temporal relations between them. Hence, a simplified representation may be used.

Definition 3. (vertical support) *Given an e-sequence* $\mathcal{S} = \{s_1, \ldots, s_n\}$ *and a temporal relation* $\mathcal{R} =< e_i, e_j, r >$ *defined by event labels* $e_i, e_j \in \Sigma$*, with* $r \in \mathcal{I}$*, the* vertical support *of* $\mathcal{R}$ *is defined as the number of e-sequences where event labels* e_i, e_j *occur with relation* r*.*

While there can be multiple occurrences of $\mathcal{R}$ in the same e-sequence, the relation is counted only once.

Let function $occ(\cdot)$ indicate a single occurrence of a temporal relation in an e-sequence, such that $occ(\mathcal{R}, \mathcal{S}) = 1$ if $\mathcal{R}$ occurs in $\mathcal{S}$, and 0 otherwise. We define a frequency function $\mathcal{F} : [0, |\mathcal{D}|] \rightarrow [0, 1]$ that computes the *relative* vertical support of a temporal relation $\mathcal{R}$ in an e-sequence database $\mathcal{D}$ as follows:

$$\mathcal{F}(\mathcal{R}) = \frac{1}{|\mathcal{D}|} \sum_{\mathcal{S}_i \in \mathcal{D}}^{|\mathcal{D}|} occ(\mathcal{R}, \mathcal{S}_i) \ .$$

Definition 4. (horizontal support) *Given an e-sequence* $\mathcal{S}$ *and a temporal relation* $\mathcal{R}$*, the* horizontal support *of* $\mathcal{R}$ *is defined as the number of occurrences of* $\mathcal{R}$ *in* $\mathcal{S}$*.*

For horizontal support, multiple occurrences of $\mathcal{R}$ in the same e-sequence are counted.

Problem 5. (**relation-preserving embedding**) Given an e-sequence $\mathcal{S}$ with u being the total number of temporal relations in $\mathcal{S}$, define a mapping function to an embedding vector space $\mathbb{R}^d$, with $d \leq u$ such that $f : \mathcal{S} \longrightarrow \mathbb{R}^d$, where the underlying temporal relations in $\mathcal{S}$ are preserved.

By construction, the proposed embedding space achieves substantially scalable solutions to the problems of **clustering** and **classification of e-sequences**.

3 Z-Embedding: a spectral embedding representation of event interval sequences

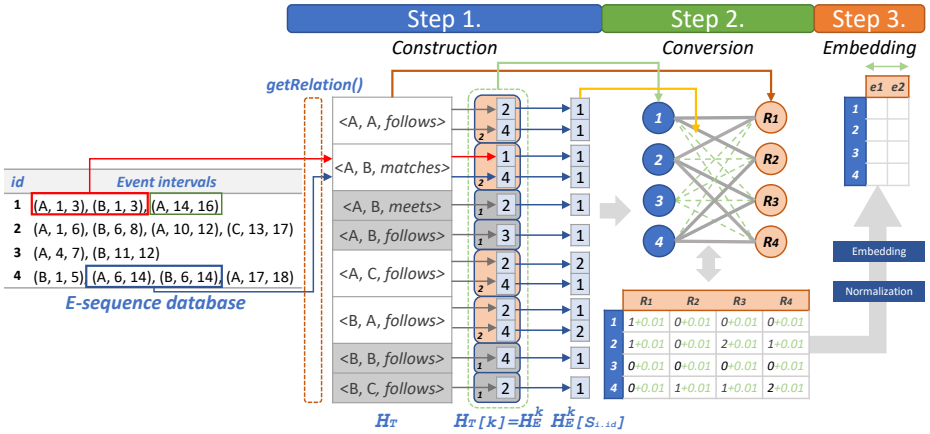

Fig. 3. An example of the process of `Z-Embedding` with the parameters {`minSup` : 0.5, `maxSup` : 1, `gap` : 0.5.}

`Z-Embedding` is an efficient three-step framework for converting an e-sequence database into a spectral embedding vector space representation, where important structural information regarding the temporal relations in the e-sequences is preserved by pruning techniques, hence facilitating scalable clustering and classification. The first two steps of the framework convert the original e-sequence database into a bipartite graph, while at the third step, the bipartite graph is converted into a spectral embedding space. The final space representation can then be readily used by off-the-shelf clustering or classification algorithms. These steps, also outlined in Figure 3 and Algorithm 1, are described below:

1. **Construction of `G-HashTable`**: This is a data structure that efficiently stores the information needed to create a bipartite graph after scanning an e-sequence database. We can apply various pruning processes based on temporal relations to the table for better graph representation.
2. **Conversion to a bipartite graph**: The pruned table is converted to a weighted bipartite graph with two vertex sets of e-sequences and temporal relations. The bipartite graph is represented as a form of a bi-adjacency matrix. We represent the bipartite graph with the two interestingness factors defined in Section 2, i.e., *vertical support* and *horizontal support* [18]. We use vertical support as a pruning factor because it is a measure of how prevalent the temporal relation is across the entire database, while horizontal support is used as a weight of the edge of the graph since it represents the strength of a specific temporal relation in different e-sequences.

3. **Spectral embedding of the bipartite graph**: After generating the bi-adjacency matrix, the feature vector of each e-sequence is generated through regularization and singular value decomposition, hence reducing the complexity and dimensionality of the e-sequences.

Since `Z-Embedding` results in numerical feature vector representation of e-sequences, we can apply a wide range of classification and clustering algorithms compared to previous distance-based (e.g., *Artemis*, *IBSM*) and non-numerical-feature-based methods (e.g., *STIFE*).

Algorithm 1: Z-Embedding

Data: $\mathcal{D}$: E-sequence database, d: dimension factor
constraints: predefined constraints {`minSup, maxSup, gap`}
Result: U: Row embedding of regularized bi-adjacency matrix

// Step 1: Construction of `G-HashTable`
$\mathcal{H}_T = \{\}$;
for $\mathcal{S}_i \in \mathcal{D}$ **do**
  for $s_a, s_b[s_a < s_b] \in \mathcal{S}_i$ **do**
    $r \leftarrow getRelation(s_a, s_b, constraints.gap)$;
    if $r \neq None$ **then**
      $\mathcal{R} \leftarrow (s_a.e, s_b.e, r)$;
      if $\mathcal{R} \notin \mathcal{H}_T$ **then**
        $\mathcal{H}_T.index(\mathcal{R})$;
      if $S_i.id \notin \mathcal{H}_T[\mathcal{R}]$ **then**
        $\mathcal{H}_T[\mathcal{R}].index(\mathcal{S}_i.id)$;
      $\mathcal{H}_T[\mathcal{R}][\mathcal{S}_i.id].addHorizontalSupport()$;
for $\mathcal{R}_k \in \mathcal{H}_T$ **do**
  if $\mathcal{F}(\mathcal{R}) < constraints.minSup \vee \mathcal{F}(\mathcal{R}) > constraints.maxSup$ **then**
    remove $\mathcal{H}_T[\mathcal{R}_k]$
// Step 2: Conversion to a bipartite graph
$B = 0^{|\mathcal{D}| \times |\mathcal{H}_T|}$;
for $\mathcal{R}_j \in \mathcal{H}_T$ **do**
  for $\mathcal{S}_i.id \in \mathcal{H}_T[\mathcal{R}_k]$ **do**
    $B[\mathcal{S}_i.id][hash(\mathcal{R}_k, |H_T|)] = \mathcal{H}_T[\mathcal{R}_k][\mathcal{S}_i.id]$;
// Step 3: Spectral embedding of the bipartite graph
$B_S = spectralEmbedding(B, d)$;
return B_S

3.1 Construction of `G-HashTable`

`G-HashTable` is a hash table composed of three layers constructed from the e-sequence database for facilitating its conversion to a bipartite graph. It efficiently

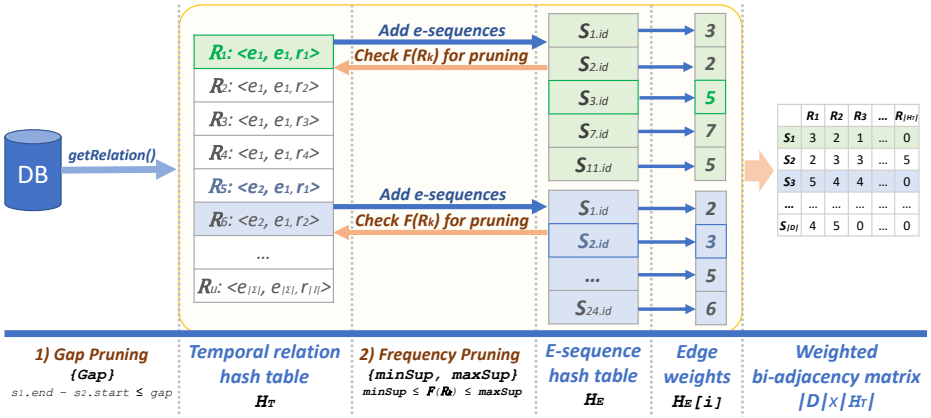

Fig. 4. An instantiation of `G-HashTable`.

maintains all information for the conversion and occurrence-based pruning by scanning temporal relations only once in the database. There are two main steps to creating the hash table: (1) construction step (blue arrows in Fig. 4), (2) pruning step (orange arrows in Fig. 4).

Construction step. First, we traverse all event intervals in the e-sequence database in chronological order. For target event interval s_a, we make a pair with all event intervals s_b that occur after s_a. Thereafter, we check the temporal relation between event intervals s_a and s_b (lines 1-5, Algorithm 1). Then, a temporal relation between them, $\mathcal{R}_k =< s_a.e, s_b.e, r >$, with $r \in \mathcal{I}$ is formed and stored as a key in the first hash table $\mathcal{H}_T$, which we call *temporal relation hash table* (lines 6-9). Whenever a relation $\mathcal{R}_k$ is found, we identify the e-sequence id containing it, and use it as a key in the second hash table $\mathcal{H}_E$, called *e-sequence hash table* (lines 10-11). We note that each record $\mathcal{H}_T[\mathcal{R}_k] \in \mathcal{H}_T$ is mapped to its respective e-sequence hash table, denoted as $\mathcal{H}_E^k$. The keys of this hash table are the e-sequence ids where $\mathcal{R}_k$ occurs, while the values are the edge weights of the bipartite graph quantifying the occurrence $\mathcal{R}_k$ in the e-sequence. When we firstly create a specific key, we set the value in $\mathcal{H}_E^k$ equal to one, which corresponds to the horizontal support of the temporal relation in the e-sequence. Thus, if the same temporal relation $\mathcal{R}_k$ occurs more than once in the same e-sequence $\mathcal{S}_i$, we add the count to $\mathcal{H}_E^k[i]$ to update its horizontal support (line 12).

Pruning step. The pruning step helps limit the unnecessary formation of relations and helping graph only to represent necessary information; This step consists of two sub-steps, which occur at different times:

1. **Gap pruning:** A gap constraint limits the maximum distance of *follows* relations between intervals. This pruning process eliminates unnecessary relations that occur just because they are far apart rather than having

a meaningful relation. The gap is checked when checking the temporal relation while scanning the database (line 5, Algorithm 1). We receive the gap constraint with a value in the range [0, 1], meaning the ratio of the average time duration of e-sequences and prune the *follows* relations having a distance above that ratio.

2. **Frequency pruning:** Frequency pruning is a step of removing temporal relations $\mathcal{R}_k$, whose relative vertical supports $\mathcal{F}(\mathcal{R}_k)$ are below or above the predefined criteria after the table is completely formed (lines 13-15). To do this, we impose the following two constraints:
   - Minimum support constraint: corresponding to the minimum occurrence frequency of each temporal relation. This helps increase the cluster's purity by limiting the small size temporal relations that can be different within a cluster.
   - Maximum support constraint: corresponding to the maximum occurrence frequency of each temporal relation. This limits the temporal relations spanning almost all e-sequences, allowing the embedding space to represent the e-sequence space holistically.

Example. Consider an e-sequence database of size 4 (Figure 3). For this example, we will use the following parameter settings: `minSup` : 0.5, `maxSup` : 1, `gap` : 0.5. Hence, we need to find relation pairs with absolute vertical supports from 2 to 4. Moreover, the gap constraint of 0.5 implies that the longest span of a *follows* relation can be at most half the average time length of all e-sequences, which is $\frac{16+17+12+18}{4} \times 0.5 = 7.875$. First, we scan the database to get all temporal relations, which is accomplished by checking the temporal relations between the event intervals in the database. In this example, we see that (A, 1, 3), (B, 1, 3) in the first e-sequence and (A, 6, 14), (B, 6, 14) in the fourth e-sequence form temporal relation: $< A, B, matches >$. Then, we place the relation as the key for the first layer of the hash table $\mathcal{H}_T$. After that, since the same temporal relation occurs in both the first and fourth e-sequences, we can store their ids into the second layer along with their corresponding vertical supports (left square boxes in the *e-sequence hash table* $\mathcal{H}_E$). Finally, we compute the horizontal support by counting how often the temporal relation has occurred in each stored e-sequence ($\{1, 4\}$ in the example). We only have a single horizontal occurrence in both e-sequences. Hence we add ones to the values of the *e-sequence hash table* (right square boxes). The gap constraint pruning is applied together with the `G-HashTable` construction. Actually, (A, 1, 3) and (A, 14, 16) in the first e-sequence must have formed a *follows* relation without the gap constraint. However, since the distance between the two event intervals is 11 (> 7.875), we skip creating a record in the `G-HashTable`. The same pruning holds for the fourth e-sequence and event intervals (B, 1, 5), (A, 17, 18) having a distance equal to 12. After constructing the `G-HashTable`, frequency pruning is performed by applying {`minSup, maxSup`}. Since $minSup = 0.5$ (or support count of 2), temporal relations with vertical support equal to 1 are subsequently excluded from the first layer of the table (gray triplets in $\mathcal{H}_T$ in the example).

The advantage of `G-HashTable` is that we can easily consider two types of frequencies and apply pruning techniques by scanning the e-sequence database only once. Moreover, we can directly convert the table to its corresponding bi-adjacency matrix weighted by the frequencies in the e-sequences. All that is required is to scan the database once and scan the first layer of the table to apply pruning and scan the first and second layers of the table to convert it into the bi-adjacency matrix.

Hence, given an e-sequence database $\mathcal{D} = \{\mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{D}|}\}$, the set of possible relations $\mathcal{I}$, and the alphabet of event labels Σ, the time complexity for creating the bi-adjacency matrix is quadratic in the worst case as follows:

$$(\sum_{\mathcal{S}_i \in \mathcal{D}}^{|\mathcal{D}|} |\mathcal{S}_i|^2 \times |\mathcal{I}|) + (|\Sigma|^2 \times |\mathcal{I}|) + (|\Sigma|^2 \times |\mathcal{I}| \times |\mathcal{D}|) \ .$$

3.2 Conversion to a weighted bipartite graph

In this step, we use the notion of a weighted bipartite graph.

Definition 6. (bipartite graph) *A bipartite graph $G = \{U, V, E\}$ is a special form of a graph, having vertices divided into two disjoint sets U and V, meaning that $U \cap V = \emptyset$, and a set of edges $E = \{e_{u,v} | u \in U, v \in V\}$.*

A bipartite graph consists of edges that can only lead from the vertex set U to the other vertex set V, while vertices belonging to the same set cannot be connected. A *weighted bipartite graph* is trivially an extension of G, where each $e_{u,v} \in E$ equals to the corresponding edge weight between u and v, or to 0 if no edge exists between u and v.

After the construction and pruning steps resulting into `G-HashTable`, we create the corresponding weighted bipartite graph by directly using each layer of the `G-HashTable`. The temporal relations in the first layer of `G-HashTable` are used as the right-hand side nodes, while the e-sequence ids of the second layer are used as the left-side nodes. Furthermore, the edges are created to link each e-sequence id (left-hand side nodes) to the corresponding temporal relations (right-hand side nodes) it contains. Horizontal supports are used as weights (having applied {`minSup, maxSup`} thresholds). The resulting graph is a weighted bipartite graph $G = \{U, V, E\}$, with

$$\begin{aligned}
&U : \{i \mid \mathcal{S}_i \in \mathcal{D}, i \in [1, |\mathcal{D}|]\},\\
&V : \{\mathcal{H}_T.keys \mid \forall \mathcal{R} \in \mathcal{H}_T : minSup < \mathcal{F}(\mathcal{R}) < maxSup\}\} \text{ and}\\
&E : \{e_{i,j} = \mathcal{H}^i_E[j] \ \mid i \in [1, |\mathcal{D}|], j \in \mathcal{H}^i_E\}.
\end{aligned}$$

Using G, we construct its bi-adjacency matrix B (lines 17-20, Algorithm 1). The bi-adjacency matrix $B \in \mathbb{R}^{|U| \times |V|}$ is a two-dimensional matrix representing G, with the dimensions corresponding to vertex sets U and V, and edges between sets U and V are defined as the elements of the matrix, with

$$B_{u,v} = \begin{cases} e_{u,v} > 0, & \text{if and only if } e_{u,v} \in E \\ 0, & \text{otherwise} \end{cases}$$

A bi-adjacency matrix is computationally efficient as it reduces the matrix size from $|U+V| \times |U+V|$ to $|U| \times |V|$, while storing the same information. In Figure 4, we see an example of a conversion from the set of temporal relations to a bipartite graph G and its corresponding weighted bi-adjacency matrix B.

Example. After the construction and pruning step, we have four temporal relations $\{< A, A, \mathit{follows} >, < A, B, \mathit{matches} >, < A, C, \mathit{follows} >, < B, A, \mathit{follows} >\}$ and four e-sequence ids {1, 2, 3, 4} that meet all the constraints. Then we can create a 4×4 bi-adjacency matrix and fill the values of the third layers of the `G-HashTable` as key of the second layer, key of the first layer in the matrix shown in Step 2 in Figure 3. For example, since the horizontal support of pair $< B, A, \mathit{follows} >$ and e-sequence 4 is 2, we can insert the value 2 into the matrix with key $\{< B, A, \mathit{follows} >, 4\}$, which is the right bottom value in the matrix. If no edge occurs between an e-sequence and a relation pair, we can set that value to zero by following the definition of the adjacency matrix. The third e-sequence will have all zeros in the matrix since all of its relations are pruned.

3.3 Spectral embedding of `Z-Embedding`

Algorithm 2: spectralEmbedding

Data: B: a bi-adjacency matrix of intervals where $B \in \mathbb{R}^{|U| \times |V|}$
d: dimension factor
Result: U: Embedding of the rows

1 $B_R = B + \alpha * 1^{|U| \times |V|}$
2 $N_B^{UR} = D_1^{-\frac{1}{2}} B_R D_2^{-\frac{1}{2}}$
3 calculate SVD $N_B^{UR} = M \Sigma W^T$
4 pick leading d singular values and corresponding d columns from M
5 **return** $M[: U, : d]$

After constructing the bipartite graph and its bi-adjacency matrix, we proceed with defining a reduced-rank spectral embedding [24]. First, we apply regularization with a regularization factor α to ensure noise and outlier robustness of the spectral embedding. The factor α is determined by prior knowledge based on the properties of the datasets. We used the most recent technique introduced in [4], which is adding a constant α equally to all elements of the bi-adjacency matrix (line 1, Algorithm 2). From a graph perspective, this means adding small-weight edges to every pair of nodes between the sets (green edges in Fig. 3).

Next, using the bi-adjacency matrix we create the *normalized Laplacian matrix* N_B. We only calculate N_B^{UR}, the top-right part of N_B (line 2).

Definition 7. (Normalized Laplacian matrix) *We define a normalized Laplacian matrix N of a graph as $N = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I_{|U|} - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where $D_U \in \mathbb{R}^{|U| \times |U|}$ is a diagonal degree matrix where $D_{ii} = deg(u_i)$, with $i \in [1, |U|]$.*

Note that we can only use the bi-adjacency matrix part (top-right), since N can be expressed as a bi-adjacency matrix as follows:

$$N = \begin{bmatrix} I_{|U|} & -D_U^{-\frac{1}{2}} B D_V^{-\frac{1}{2}} \\ -D_V^{-\frac{1}{2}} B^T D_U^{-\frac{1}{2}} & I_{|V|} \end{bmatrix} \quad (1)$$

The normalized Laplacian matrix provides an approximate solution for finding the sparsest cut of the graph, providing a good graph partition [19].

The next step is to define the spectral embedding space of N_B^{UR}, hence reducing the horizontal dimension of the matrix, which can have a maximum size of $|\Sigma|^2 * |\mathcal{I}|$, to create reduced-size feature vectors that can be processed at a faster speed. The spectral embedding space is achieved by constructing a new embedding space and obtaining the leading eigenvectors of the adjacency matrix. Since the bi-adjacency matrix is not square, we apply SVD as an equivalent process of eigendecomposition [16]. Then we sort the singular values and choose the d leading values, where $d \leq min(|U|, |V|)$, and the corresponding columns from M. The target dimension parameter k is set based on prior knowledge and the dataset properties. Finally, we return the selected d columns of M (size $U \times d$), which defines the spectral embedding space of each e-sequence.

The intuition behind these three steps is that pruning in the spectral space will result in e-sequences of the same class label having similar but unique distributions of pairwise relations.

4 Experiments

We demonstrated the applicability of the `Z-Embedding` representation on five real-world datasets for clustering and classification, and compared it against two state-of-the-art competitors for the task of clustering and three for classification. For **repeatability** purposes, our datasets and code can be found on github[1].

4.1 Setup

Datasets. We used five public datasets collected from different application domains. Table 1 summarizes the properties of datasets. Detailed information for each dataset can be found in earlier works (e.g., [7]).

Competitor methods. We demonstrated the runtime efficiency of `Z-Embedding` and its applicability to the clustering tasks and classification tasks. More concretely, for clustering, we benchmarked k-means and k-medoids under the Euclidean distance in the `Z-Embedding` space, and compared them against using two alternative state-of-the-art distance functions, *Artemis* and *IBSM*. Moreover, for classification, we benchmarked four different classifiers using the `Z-Embedding` feature space, i.e., 1-Nearest Neighbor (1-NN), Random Forests (RF), Support

[1] `https://github.com/zedshape/zembedding`

Table 1. A summary of the properties of the real-world datasets.

Dataset	# of e-seq.	# of event labels	# of event intervals	Avg. interval length	Avg. e-seq. length	# of unique temp. rel.	# of total temp. rel.
BLOCKS	210	8	1,207	5.75	54.13	174	3,245
PIONEER	160	92	8,949	55.93	57.19	26,429	252,986
CONTEXT	240	54	19,355	80.65	191.85	6,723	804,504
SKATING	530	41	23,202	43.78	1,916.08	4,844	516,272
HEPATITIS	498	63	53,921	108.28	3,193.55	20,865	3,785,167

Vector Machine (SVM) with the Radial Basis Function (RBF) kernel (SVM_RBF), and SVM with the 3-degree polynomial kernel (SVM_Poly). These were compared against 1-NN using *Artemis* and *IBSM*. For completeness, we additionally compared against *STIFE*, a RF feature-based classifier for e-sequences. All clustering and classification algorithms were implemented using the scikit-learn library.

4.2 Results

All algorithms were implemented in Python 3.7 and run on an Ubuntu 18.10 system with Intel i7-8700 CPU at 3.20GHz and 32GB main memory. All results contain the average values of 10-fold cross-validation (for classification) and 100 trials (for clustering). If the algorithm required hyperparameters, we followed the parameter setup defined by the authors of each paper for a fair comparison. For the dimension factor d for spectral embedding, we chose $d = 4$ for BLOCKS dataset as it has comparably smaller in terms of the number of temporal relations compared to other datasets (Table 1), while for the rest of the datasets we set $d = 8$. Throughout this process, the resulting feature vectors provided a compressed version of the original space by almost 99% for all datasets, which has contributed to the high computation speedups obtained. Using `Z-Embedding` , we could achieve speedups of up to a factor of 292 compared to the competitors[2].

Clustering results. We set the expected number of clusters to the actual number of class labels in the dataset, and computed the total runtime and purity values required for all the algorithms. Since *Artemis* is only calculating distances, k-means was inapplicable. K-medoids was generally faster than k-means because it could be run after pre-calculating the pairwise distances between the data e-sequences. To construct the embedding space for `Z-Embedding`, we used the same regularization factor, $\alpha = 0.001$, for every dataset.

Table 2 shows the results in terms of clustering purity and runtime. For each method, we set a one-hour time limit to its runtime. We firstly applied each algorithm to create the feature vectors, and then k-means and k-medoids were applied, respectively. In terms of runtime, `Z-Embedding` was faster in both

[2] This is even an underestimate as for the cases where competitors that did not finish within the one-hour execution time limit, our approach is at least 300 times faster.

Table 2. Clustering results for `Z-Embedding` and all competitors in terms of clustering purity (%) and runtime (seconds).

Dataset	*Artemis*		*IBSM*				`Z-Embedding`			
	K-medoids		K-medoids		K-means		K-medoids		K-means	
	Purity	Time	Purity	Time	Purity	Time	Purity	Time	Purity	Time
BLOCKS	85.62	1.20	95.30	0.71	99.09	10.57	93.81	**0.02**	99.82	0.04
PIONEER	66.13	15.64	63.94	4.41	64.09	74.13	74.75	**0.89**	83.12	0.91
CONTEXT	65.13	122.23	75.22	5.19	82.66	204.82	77.54	**1.99**	82.36	2.02
SKATING	36.52	180.48	70.21	286.10	-	>1h	62.45	**1.48**	74.40	1.52
HEPATITIS	-	>1h	67.91	444.77	-	>1h	71.70	**9.60**	70.08	9.63

cases compared to two competitors. In particular, *Artemis* did not complete the calculation within an hour on the HEPATITIS dataset. *IBSM* showed deficient runtime performance for the datasets with long e-sequences, such as SKATING or HEPATITIS. Specifically, when k-medoids was used on SKATING, the speed was even slower than that of *Artemis*. Moreover, k-means could not complete within an hour on SKATING and HEPATITIS, while `Z-Embedding` with k-means completed in 1.52 seconds on SKATING and 9.63 seconds on HEPATITIS.

In terms of purity, `Z-Embedding` also showed remarkable results. In the k-medoids trials, *Artemis* had the lowest purity values on all data sets except for PIONEER. *IBSM* showed the highest purity only on SKATING with k-medoids, but it was about 193 times slower than `Z-Embedding`. On the other datasets, `Z-Embedding` showed the fastest runtime performance and achieved the highest purity. In the k-means experiment, `Z-Embedding` showed the highest purity values, except for CONTEXT. *IBSM* led by a slight difference of 0.3 percent on CONTEXT but was also about 101 times slower than `Z-Embedding`.

Classification results For each competitor method, we used the classifiers suggested by the authors in the corresponding papers. Since *Artemis* and *IBSM* are distance-based algorithms, the number of applicable algorithms is highly limited. Therefore, for these two competitors, the 1-NN classifier was applied. On the other hand, since *STIFE* generates non-numeric feature vectors, distance-based algorithms cannot be applied, and in this case, RF was applied. For *STIFE*, we applied the recommended optimal parameters [2]. In order to adjust the parameters of `Z-Embedding` for each dataset, we performed a grid search on 1-NN classification accuracy within the range of [0, 1] for each of the three parameters {`maxSup, minSup, gap`}, in increments of 0.1. The top 10 parameter settings and the experimental results are available in the supplementary materials.

Unlike existing algorithms, `Z-Embedding` can apply a wide range of algorithms as it forms numeric feature vectors. In this experiment, we applied the ones that previous methods used, such as 1-NN and RF, and we also ran two SVM with RBF kernel and polynomial kernel. Table 3 shows the classification accuracy and runtime for each competitor method, while Table 4 shows the results for `Z-Embedding`. 1-NN under *Artemis* had the longest runtime and lowest accuracy

Table 3. Classification results for all competitors in terms of classification accuracy (%) and runtime (seconds).

Dataset	*Artemis*		*IBSM*		*STIFE*	
	1-NN		1-NN		RF	
	Acc	Time	Acc	Time	Acc	Time
BLOCKS	98.57	1.43	100	0.77	100	2.96
PIONEER	95.00	19.27	93.75	4.43	98.75	8.51
CONTEXT	92.50	130.22	97.08	5.32	98.33	12.1
SKATING	84.92	208.79	97.74	286.24	96.42	21.4
HEPATITIS	-	>1h	77.91	445.83	82.13	83.7

Table 4. Classification results for `Z-Embedding` in terms of classification accuracy (%) and runtime (seconds).

Dataset	`Z-Embedding`										
	Constraints			1-NN		RF		SVM_RBF		SVM_Poly	
	`minSup`	`maxSup`	`gap`	Acc	Time	Acc	Time	Acc	Time	Acc	Time
BLOCKS	0.0	0.4	0.0	100	**0.02**	100	0.12	100	**0.02**	100	**0.02**
PIONEER	0.0	0.7	0.1	100	**1.49**	100	1.62	100	1.50	100	**1.49**
CONTEXT	0.4	0.5	0.2	95.00	**1.35**	96.25	1.46	97.50	1.36	97.08	1.36
SKATING	0.5	0.6	0.1	91.32	**0.98**	92.07	1.10	93.58	0.99	92.45	0.99
HEPATITIS	0.0	1.0	0.1	76.30	10.83	82.13	11.27	83.73	**10.82**	83.34	11.04

for all datasets except for PIONEER, while on HEPATITIS it failed to complete within the 1-hour runtime limit. On the other hand, *IBSM* achieved the best performance on SKATING, but it is 13 times slower than *STIFE* and up to 292 times slower than `Z-Embedding` . Finally, *STIFE* was the algorithm with the highest speed and accuracy performance (except for SKATING) among the other competitors. It even achieved better performance than `Z-Embedding` on CONTEXT and SKATING, but it was about up to 9 times slower than `Z-Embedding` on CONTEXT, and 21 times on SKATING.

5 Conclusion

We proposed a novel representation of event interval sequences using a bipartite graph for efficient clustering and classification. We benchmarked our representation on five real-world datasets against several competitor algorithms. Our experimental benchmarks showed that the proposed spectral embedding representation can achieve substantially lower runtimes compared to earlier competitors and even higher values of purity (for clustering) and classification accuracy (for classification) than some of its competitors. Future work includes extending the bipartite graph representation to tripartite or higher multipartite by calculating higher orders of temporal relations, investigating the usage of our framework for providing scalable solutions to other machine learning problems, and also exploring alternative link-analysis ranking methods such as rooted PageRank.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. CACM **26**(11), 832–843 (1983)
2. Bornemann, L., Lecerf, J., Papapetrou, P.: Stife: A framework for feature-based classification of sequences of temporal intervals. In: DS. pp. 85–100. Springer (2016)
3. Chaudhuri, K., Chung, F., Tsiatas, A.: Spectral clustering of graphs with general degrees in the extended planted partition model. In: COLT. pp. 35–1 (2012)
4. De Lara, N., Bonald, T.: Spectral embedding of regularized block models. In: ICLR (2020)
5. Joseph, A., Yu, B., et al.: Impact of regularization on spectral clustering. ANN STAT **44**(4), 1765–1791 (2016)
6. Kostakis, O., Gionis, A.: On mining temporal patterns in dynamic graphs, and other unrelated problems. In: COMPLEX NETWORKS. pp. 516–527. Springer (2017)
7. Kostakis, O., Papapetrou, P.: On searching and indexing sequences of temporal intervals. DMKD **31**(3), 809–850 (2017)
8. Kotsifakos, A., Papapetrou, P., Athitsos, V.: Ibsm: Interval-based sequence matching. In: SDM. pp. 596–604. SIAM (2013)
9. Kunegis, J.: Exploiting the structure of bipartite graphs for algebraic and spectral graph theory applications. Internet Mathematics **11**(3), 201–321 (2015)
10. Lam, H.T., Mörchen, F., Fradkin, D., Calders, T.: Mining compressing sequential patterns. SADM **7**(1), 34–52 (2014)
11. Liu, L., Wang, S., Hu, B., Qiong, Q., Wen, J., Rosenblum, D.S.: Learning structures of interval-based bayesian networks in probabilistic generative model for human complex activity recognition. PR **81**, 545–561 (2018)
12. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: NIPS. pp. 849–856 (2002)
13. Perera, D., Kay, J., Koprinska, I., Yacef, K., Zaïane, O.R.: Clustering and sequential pattern mining of online collaborative learning data. TKDE **21**(6), 759–772 (2008)
14. Pissinou, N., Radev, I., Makki, K.: Spatio-temporal modeling in video and multimedia geographic information systems. GeoInformatica **5**(4), 375–409 (2001)
15. Qin, T., Rohe, K.: Regularized spectral clustering under the degree-corrected stochastic blockmodel. In: NIPS. pp. 3120–3128 (2013)
16. Ramasamy, D., Madhow, U.: Compressive spectral embedding: sidestepping the svd. In: NIPS. pp. 550–558 (2015)
17. Schmidt, M., Palm, G., Schwenker, F.: Spectral graph features for the classification of graphs and graph sequences. CompStat **29**(1-2), 65–80 (2014)
18. Sheetrit, E., Nissim, N., Klimov, D., Shahar, Y.: Temporal probabilistic profiles for sepsis prediction in the icu. In: KDD. pp. 2961–2969 (2019)
19. Shi, J., Malik, J.: Normalized cuts and image segmentation. TPAMI **22**(8), 888–905 (2000)
20. Von Luxburg, U.: A tutorial on spectral clustering. Statistics and computing **17**(4), 395–416 (2007)
21. Von Luxburg, U., Belkin, M., Bousquet, O.: Consistency of spectral clustering. ANN STAT pp. 555–586 (2008)
22. Wang, J., Han, J.: Bide: Efficient mining of frequent closed sequences. In: ICDE. p. 79. IEEE (2004)
23. Zhang, Y., Rohe, K.: Understanding regularized spectral clustering via graph conductance. In: NIPS. pp. 10631–10640 (2018)
24. Zhou, Z., Amini, A.A.: Analysis of spectral clustering algorithms for community detection: the general bipartite setting. JMLR **20**(47), 1–47 (2019)