Data Min Knowl Disc (2017) 31:1840–1871
DOI 10.1007/s10618-017-0515-0

Discovering recurring activity in temporal networks

Orestis Kostakis[1] · **Nikolaj Tatti**[2] · **Aristides Gionis**[2]

Received: 29 February 2016 / Accepted: 22 May 2017 / Published online: 17 June 2017

Abstract Recent advances in data-acquisition technologies have equipped team coaches and sports analysts with the capability of collecting and analyzing detailed data of team activity in the field. It is now possible to monitor a sports event and record information regarding the position of the players in the field, passing the ball, coordinated moves, and so on. In this paper we propose a new method to analyze such team activity data. Our goal is to segment the overall activity stream into a sequence of potentially recurrent modes, which reflect different strategies adopted by a team, and thus, help to analyze and understand team tactics. We model team activity data as a temporal network, that is, a sequence of time-stamped edges that capture interactions between players. We then formulate the problem of identifying a small number of team modes and segmenting the overall timespan so that each segment can be mapped to one of the team modes; hence the set of modes summarizes the overall team activity. We prove that the resulting optimization problem is NP-hard, and we discuss its properties. We then present a number of different algorithms for solving the problem, including an approximation algorithm that is practical only for one mode, as well as heuristic methods based on iterative and greedy approaches. We benchmark the performance of our algorithms on real and synthetic datasets. Of all methods, the iterative

The work was carried out while the author was at Aalto University, Espoo, Finland.

✉ Orestis Kostakis
orkostak@microsoft.com

Nikolaj Tatti
Nikolaj.Tatti@aalto.fi

Aristides Gionis
Aristides.Gionis@aalto.fi

[1] Microsoft Corporation, Redmond, WA, USA

[2] HIIT, Aalto University, Espoo, Finland

algorithm provides the best combination of performance and running time. We demonstrate practical examples of the insights provided by our algorithms when mining real sports-activity data. In addition, we show the applicability of our algorithms on other types of data, such as social networks.

Keywords Temporal networks · Dynamic graphs · Summarising · Segmentation · Sports analytics · Basketball · Football · Handball · Social networks

1 Introduction

A key factor of success in most team sports, perhaps even greater than the athletic abilities and performance of individual players, is the team strategy and game tactics: which lineup to be used for a given game, how the team is set in the field, what combinations the players are using to coordinate with each other (e.g., on passing the ball), and so on. Some of the most successful teams in the sports history are known not only for their winning achievements but also for revolutionizing the tactical theory of their respective sport. Famous examples include the "tiki-taka" passing style of modern-day Barcelona football team, or the "neutral trap zone" popularized by New Jersey Devils ice hockey team during the "dead puck era". Similarly, the "triangle offense" popularized by Phil Jackson allowed his teams to win 11 NBA championships.

New technologies introduced in the field of sports analytics support the collection of detailed high-frequency data, such as x–y positioning of the players and x–y–z positioning of the ball in the field. Analysis of this data can provide in-depth statistics, which can offer valuable insights to the fans, or can help teams dissect and develop their game, or comprehend and counter that of their opponents. In addition to reporting simple statistics (possession of the ball, distance covered, percentage of successful passes, etc.), the collected data can form higher-level representations that can be used to analyze and understand team tactics. As an example used in this paper, one can view one game of a team (or a series of games) as a *temporal network*, where nodes represent individual players and every pass between two players is represented as a time-stamped edge.

In this paper we consider data recording the activity of a team in a game, or a series of games. We assume that the data represent interactions between team players, such as, passing the ball, moving near to each other, or being in the field at the same time. We model this activity data as a *temporal network*, that is, a sequence of time-stamped edges between nodes. We also assume that the team activity can be segmented into a sequence of *modes*, which reflect different strategies adopted by the team. Examples of such modes include team configurations aimed to achieve a target result (e.g., playing offensive or defensive), combinations of players who are able to collaborate well, systems that involve a subset of players moving and passing in a certain way, and so on. Our goal in this paper is to develop methods that discover recurring modes in a temporal network of team activity. Such an analysis can be a valuable tool for providing insights into the strategy of a team.

Formulated in computer-science terms, we consider the problem of *summarizing temporal networks*. We view temporal networks as a sequence of edges with timestamps, which record the time that network entities interact with each other. Our

underlying assumption is the network operates in a small number of different *modes*, or *summaries*, and that the overall life-span of the network can be seen as a transition between those modes. The *time segment* that the network operates in the same mode is called *session*. Viewed in this light, at a high-level our work is related to the areas of *sequence segmentation* (Bellman 1961), as well as *latent modeling* (Heinen 1996). However, in the context of temporal networks, the problem of discovering recurring modes of operation is novel.

In addition to applications in sports analytics, our problem formulation has a number of different applications. For instance, an operator of a mobile communications network would be interested in identifying the different modes under which the network functions, and what is the best description of each one of those modes. Alternatively, one can apply our method to summarize a social network by the means of identifying the periods that the communication patterns between the members of the social network change, and understand how those communication patterns recur over time.

We approach the problem of summarizing a temporal network from a combinatorial point of view. We seek to segment the network life-span in k intervals and map each of those intervals to a *latent graph*, chosen among a small set of h such graphs. Those h graphs correspond to the hidden summaries that we identify.

The contributions of this work are summarized below:

- We provide a formal definition of this temporal network summarization problem, which we name TEMP- NET- SUMM, and show that it is NP-hard.
- We propose and investigate several heuristics, including: an iterative algorithm that alternates between finding the k segments and the h summaries, two algorithms based on a greedy bottom–up merging approach, and a theoretical factor-2 approximation algorithm for the original problem, which is practical only for very small values of h.
- We present a quasi-linear $(1+\epsilon)$-approximation algorithm for finding the k segments, given the h summaries.
- Finally, we have implemented all the proposed methods, and we have performed extensive evaluation using synthetic datasets and sports-related datasets, obtaining meaningful insights.

The rest of this paper is organized as follows. In Sect. 2 we introduce our notation, we formally define our problem, and we discuss and prove its properties. We discuss the work that is most related to our paper in Sect. 3. In Sect. 4 we present our algorithms, including the iterative method, the theoretical factor-2 approximation algorithm, and the two variants of the bottom–up approach, while Sect. 5 is devoted to the speed up of the dynamic programming. Our experimental evaluation is presented in Sect. 6 and finally, Sect. 7 provides the conclusion and directions for future work.

2 Preliminaries and problem definition

2.1 Notation

A *temporal* network $H = (V, \mathcal{E})$ consists of a set of vertices V and a sequence of time-stamped edges $\mathcal{E} = (e_1, t_1), \ldots, (e_\ell, t_\ell)$, where $e_i \in V \times V$. Edges can appear several

times and multiple edges can share the same time stamps. For notational simplicity, we will assume that time stamps are all integers. Our model assumes that the set of vertices V remains constant, but it can be retrofitted to handle adding and removing vertices simply by taking the union of the vertices over all time-points.

Given a temporal network $H = (V, \mathcal{E})$, we define a *topology graph* $\pi(H) = (V, F)$, where F consists of edges, without time stamps, occurring at least once in $\mathcal{E}$. $\pi(H)$ is the *projection* of the temporal network. We will always refer to networks without time stamps as graphs.

Given a temporal network $H = (V, \mathcal{E})$ with ℓ edges, over its total time span $[1, n]$, and two time stamps $a < b$, we define a *segment* $H[a, b] = (V, \mathcal{F})$ by only keeping the edges within the time interval $[a, b]$, that is, $\mathcal{F} = \{(e, t) \in \mathcal{E} \mid a \leq t \leq b\}$. The definition of projection $\pi(H[a, b])$ extends to segments. Moreover, we define a k-segmentation to be a sequence of k time intervals $\langle (s_i, f_i) \rangle_{i=1}^{k}$, where (s_i, f_i) is a time interval. The segments must satisfy $s_1 = 1$, $f_k = n$, and $s_i = f_{i-1} + 1$; each interaction belongs to exactly one time interval.

For two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, we define the distance $d(G_1, G_2)$ to be the symmetric difference between the edge sets, that is, $d(G_1, G_2) = |E_1 \backslash E_2| + |E_2 \backslash E_1|$.

Given ν graphs $G_1 = (V, E_1), \ldots, G_\nu = (V, E_\nu)$, we define their *centroid* to be a new graph $C^* = (V, E^*)$ that minimizes the distance $d(X \mid G_1, \ldots, G_\nu) = \sum_{i=1}^{\nu} d(G_i, X)$, that is, $C^* = \arg\min_X d(X \mid G_1, \ldots, G_\nu)$. The centroid graph $C^* = (V, E^*)$ can be easily computing using the *majority rule*: for two vertices $u, v \in V$ we set $(u, v) \in E^*$ if and only if (u, v) is an edge that is present in the majority of the input graphs $G_1, \ldots, G_\nu$.

As already discussed, we assume that the temporal network operates in a small number of different modes, and that the overall life-span of the network can be seen as a transition between those modes. Recall also that the time segment that the network operates in the same mode is called session. Let us assume that there are h different modes and k sessions. A mode is modeled by a graph C, while a session is a segment $H[a, b]$. We expect that if the session $H[a, b]$ corresponds to the mode C, then the topology graph $\pi(H[a, b])$ resembles as much as possible the graph C; namely, the distance $d(\pi(H[a, b]), C)$ is small.

On the other hand, if different sessions $H[a_1, b_1], \ldots, H[a_\nu, b_\nu]$ correspond to the same mode C, then the best graph to represent the mode C, is the centroid C^* of the graphs $\pi(H[a_1, b_1]), \ldots, \pi(H[a_\nu, b_\nu])$. Since we are using the centroid graph C^* to *summarize* a set of different sessions of H, the graph that represents a mode is also called *network summary*, or simply *summary*.

Given a temporal network H, a k-segmentation $S = \langle (s_i, f_i) \rangle_{i=1}^{k}$, and h network summaries $\mathcal{C} = \{C_1, \ldots, C_h\}$ we define a penalty score

$$p(H, S, \mathcal{C}) = \sum_{i=1}^{k} \min_{C \in \mathcal{C}} d(\pi(H[s_i, f_i]), C), \tag{1}$$

measuring the total distance of representing each segment of S with the network summary in $\mathcal{C}$ that provides the *best fit*.

In addition, we would like to avoid trivial solutions, in which the resulting segmentation is dominated by few long segments. We achieve this by using an upper-bound constraint R on the length of the segments in our solution. Such a constraint is analogous to the Sakoe-Chiba band (Sakoe and Chiba 1971), used in dynamic time warping (DTW) distance measure between time-series.

2.2 Problem definition

We are now ready to define the problem that we address in this paper, which we call TEMP- NET- SUMM, for *temporal-network summarization*.

Problem 1 (Temp-Net-Summ) Given a temporal network H, the number of segments k, the number of summaries h, and the maximum length of a segment R, find a k-segmentation S, and a set of h summaries $\mathcal{C} = \{C_1, \ldots, C_h\}$, minimizing $p(H, S, \mathcal{C})$, such that each segment has at most length of R.

In the rest of our discussion, given a temporal network H, and parameters k, h and R, we use $p(H, k, h, R)$ to denote the value of the *optimal solution* to the TEMP- NET-SUMM problem with the corresponding input. For brevity, we will often omit R and write $p(H, k, h)$.

We provide an example instance of the problem. The temporal network H depicted in Fig. 1, has time span 6, and its projection $\pi(H[1, 6])$ is the graph depicted in Fig. 2a. An optimal segmentation of H, with $k = 4$ and $h = 2$ yields the segments: $[t_1, t_2]$, $[t_3, t_4)$, $[t_4, t_5)$, $[t_5, t_6]$. This segmentation results in the summaries depicted in Fig. 2b. Then, the temporal network can be summarized via the sequence C_1, C_2, C_1, C_2 using those two summaries. Notice that in this example $d(\pi(H[1, 2]), C_1) = 0$ and $d(\pi(H[5, 6]), C_2 = 0)$; the projections of all 4 sessions are identical to some summary. Finally, for this temporal network, $p(H, 4, 2) = 0$.

For the remainder of this paper we will use the following short hand notation: k and h for the number of segments and summaries, n for the network's time span (number of time-points), m the number of the edges in $\pi(H[1, n])$, ℓ the total number of edges. Finally, R would be the maximum allowed length for a segment.

2.3 Insights into the problem

By examining the problem, we discover that it belongs to the class of NP-complete problems. A proof for the case where $h \geq 2$ follows from a reduction of the

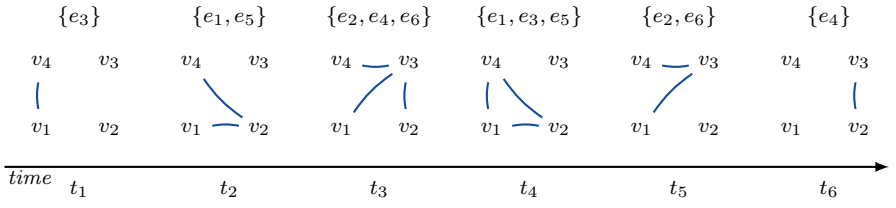

Fig. 1 Example of temporal network H, with $|V| = 4$ vertices, time span $n = 6$, $m = 6$ edges in $\pi(H[1, n])$ and $\ell = 12$ total edges

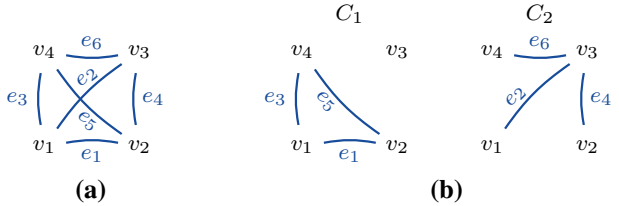

Fig. 2 The projection $\pi(H[1,6])$ and the network summaries, when $k = 4$ and $h = 2$, of the temporal network of Fig. 1. The two summaries C_1, C_2 perfectly summarize H; $p(H,4,2) = 0$, since $d(\pi(H[1,2]), C_1) = 0$ and $d(\pi(H[5,6]), C_2 = 0)$. **a** Projection. **b** Network summaries, C_1 and C_2

HYPERCUBE- SEGMENTATION problem (Kleinberg et al. 1998). The problem remains NP-hard even for $h = 1$, and we provide a more elaborate reduction.

Proposition 1 TEMP- NET- SUMM *is NP-complete, for* $h \geq 2$.

Proof The proof follows via a reduction of HYPERCUBE- SEGMENTATION problem (Kleinberg et al. 1998) to TEMP- NET- SUMM. An instance of HYPERCUBE- SEGMENTATION is a set of n graphs. For each graph, we create a time point with that graph. Then we solve TEMP- NET- SUMM with $k = n$, to obtain a solution for the first problem. Having k equal to n is a special case of TEMP- NET- SUMM, and is NP-hard. Hence, TEMP- NET- SUMM is NP-hard in the general case. NP-completeness follows from the fact that given a k-segmentation and the h summaries, we can verify the solution in polynomial time with respect to the input size. □

Proposition 2 TEMP- NET- SUMM *is NP-hard for* $h = 1$.

Proof Moved to "Appendix". □

The maximum score of any TEMP- NET- SUMM instance is km; k segments (sessions) need to be matched to summaries, each induces a score of at most m. A very naive algorithm guarantees a maximum scores of $km/2$: choose the segments and summaries arbitrarily, and if the score is greater than $km/2$, replace the summaries with their complement-graphs.

The TEMP- NET- SUMM problem satisfies the monotonicity property with respect to the number of network summaries h. All else remaining equal, as the number of summaries h increases, the score of the optimal solution may only decrease. Intuitively, given a solution to the TEMP- NET- SUMM problem for values k and h, one can easily obtain a solution for values k and $h + 1$: by selecting h out of $h + 1$ summaries to be same as before, which would yield a less or equal score.

Proposition 3 $p(H,k,h) \geq p(H,k,h+1)$

However, the monotonicity property does not hold with respect to the number of segments k, unless we allow empty segments. One may consider the case where the same pattern repeats twice; all edges appear exactly twice and in the same order relative to the others. Then for $k = 2$ and $h = 1$ the optimal score is zero; the optimal

segmentation is obtained by halving the timeline. The optimal solution of the same problem instance with $k = 3$ and $h = 1$, however, would return a non-zero score. Since all network interactions occur only twice, and none 3 times, they would be shared among one or two segments out of three. Thus, for each possible edge, if the network summary contains it, then the segments that do not contain it would contribute to the score, and vice versa.

Finally, we note that $p(H, k, k) = 0$ (when $h = k$), and the optimal solution is *any* valid k-segmentation; then each mode becomes its own summary. This observation is in line with the intuition of our problem formulation, where we would expect that the number of network summaries is significantly smaller than the number of segments, i.e., $h \ll k$.

3 Related work

Although this problem of summarizing temporal networks has not been studied before, our paper has connections with two independent lines of work: *analysis of temporal networks* and *sequence-segmentation problems*. In the first part of this section we are reviewing the main themes in these two areas, and we point out the differences and similarities with our work. In the second part we discuss other approaches from the field of Data Mining that focus on sports analytics.

Analysis of temporal networks Researchers have considered many different models of temporal networks, and the terminology includes names such as, *dynamic graphs*, *evolving graphs*, *time-varying graphs*, etc. For instance, *dynamic graph algorithms* is a classic area in graph algorithms, where the goal is to maintain graph primitives, such as, connectivity, shortest paths, spanning trees, spanners, etc., when nodes and edges are added or deleted (Eppstein et al. 1998; Henzinger and King 1999; Holm et al. 2001; Thorup 2000).

Other models consider dynamic graphs as a sequence of graph snapshots. A well-studied task in this model is the problem of tracking the evolution of communities (Asur et al. 2009; Greene et al. 2010; Sun et al. 2007). Similar snapshot-based approaches have been followed for the problem of detecting events in dynamic graphs. The research tools developed in this area include novel metrics for measuring similarity or distance between graphs (Papadimitriou et al. 2010; Gao et al. 2010; Sricharan and Das 2014), while a number of papers relies on matrix-decomposition methods (Ide and Kashima 2004). A recent line of work (Araujo et al. 2014; Shah et al. 2015) applies the Minimum Description Length (MDL) principle in order to discover instances of pre-defined graph patterns in dynamic graphs.

Another problem is that of mining dynamic graphs in order to detect evolving network processes (Mongiovi et al. 2013). In networks with time-varying node- or edge-weights, the task is to find subgraphs in a subsequence of the given snapshots, such that the subgraphs evolve *smoothly*. The found subgraphs should share a given number of edges during successive snapshots, and hence, this approach cannot be retrofitted for the purpose of our task.

Berlingerio et al. (2009) presented an algorithm for mining graph evolution rules. It is applied to social networks and makes the assumption that the graph can only grow;

edges may only be added. It may be extended to handle edge-deletions only if an edge is created and deleted at most once.

Finally, another approach to summarizing network activity is by providing the appropriate visualization techniques. Appan et al. (2006) suggested a ring-based visualization of temporal networks for discovering a predefined set of patterns. Network snapshots at different time points are depicted as concentric circles. The vertices correspond to people and they are coloured if that person is active at a given network snapshot. This approach does not provide any information the graph's edges and can only facilitate the analysis of small networks.

Clearly, all of the above works have different goals from the summarization problem presented here. In most cases, even the graph model is different. The *temporal network* model we consider is the one used by Rozenshtein et al. (2014) for the task of finding dynamic dense subgraphs, and by Kumar et al. (2015) for the task of maintaining neighborhood sizes in sliding windows. A thorough survey on *temporal networks*, discussing other problems, is provided by Holme and Saramäki (2012).

We focus on summarizing a single temporal network. In other words there is one large underlying graph. Other works such as that by Kostakis (2014) and Rayana and Akoglu (2016) focus on summarizing multiple different graphs.

Segmentation problems Our work is inspired by that of Gionis and Mannila (2003), where the goal is to discover k segments in time-series data using only h centroids. The main technical difference here is the different quality score for segments, and so we cannot use the same techniques suggested in that paper.

If we do not limit the number of centroids, or if we fix the centroids, the optimization problem becomes significantly simpler, leading to a classic segmentation problem which can be solved by a dynamic program (Bellman 1961). The running time of this program is quadratic, but efficient heuristics have been suggested by Shatkay and Zdonik (1996), Himberg et al. (2001). Finally, Guha et al. (2006) intoduce an efficient algorithm yielding $(1 + \epsilon)$ approximation guarantee. Unfortunately, we cannot use this approach as the required conditions do not hold in our settings.

In the field of sports analytics, most methods apply domain-based approaches. Hence, they are applicable to one or few sports, or it is non-trivial to make them universal. Zhong and Chang (2001) showed how to segment video based on identifying the camera angle using the court's boundaries and lines; the models for the boundaries need to be provided by a domain expert. Their approach is applicable to sports such as tennis and baseball that comprise of simple and recurring actions and alternate between similar video scenes. Extensions such as that by Pei and Chen (2003) incorporate state-transition graphs and assume that the video alternates between these states. These graphs, too, need to be given in advance by a domain expert. For snooker, Denman et al. (2003) devised methods for detecting events by discovering which object has disappeared from view; this corresponds to balls going into the pots.

Spatiotemporal analysis Another approach to segmenting the game is by using spatiotemporal analysis and in particular the trajectories of the players or the ball. For example, Wei et al. (2013) use the locations of football players in order to identify their actions and to eventually segment the game in phases such as *in-play* or *stoppage*.

For basketball matches, Perše et al. (2009) devise a method for segmenting the match in offensive actions, defensive actions, or time-outs. The goal is to eventually match each segment into a set of pre-defined in-game plays.

In addition, trajectory mining has been used for detecting formations of players; their position relative to the field, other players, the position of the ball, etc. This is useful for detecting the role of each player in field-hockey matches (Lucey et al. 2013a). Similarly, it is possible to use the team-formation data in order to gain deeper insights into a sport and answer questions such as why the home team has an advantage (Lucey et al. 2013b). Overall, due to the development of more accurate tracking systems (Hayet et al. 2005; Stensland et al. 2014; Pingali et al. 1998), spatiotemporal analysis has recently become a very popular research area for facilitating sports. For more details we refer the reader to a recent survey paper by Gudmundsson and Horton (2016).

We note, however, that trajectory clustering and analysis of moving objects are both orthogonal topics to the problem that we study in this paper, as our input data are not trajectories, but time-stamped interactions, i.e., a temporal network.

Sports Analytics Our work can be identified in the broad area of *sports analytics*. While there is no formal definition of this scientific discipline, informally, sports analytics goes beyond the simple use of statistics for evaluating team- and player performance (Travassos et al. 2013), and refers to the whole breadth of computer science with application to sports (Alamar 2013).

A big part of the work in the literature refers to *predictive analytics*, and in particular, predicting the outcome of matches; examples of such works include predicting game outcomes in association football (soccer) (Crowder et al. 2002; Hvattum and Arntzen 2010) and American football (gridiron) (Harville 1980). A significant body of work focuses on using *computer vision* and *image analysis* techniques for analyzing segments of gameplay action and classifying them into different types (Goldsberry 2012; Halvorsen et al. 2013; Kasiri-Bidhendi et al. 2015; Maheswaran et al. 2012).

Other works focus on less mainstream sports applications and analysis of data from a broader sports-data ecosystem. Examples include detecting match-fixing in tennis (Rodenberg and Feustel 2014), and height-bias among basketball referees (Gift and Rodenberg 2014). Yet in a different direction, other researchers have considered multiplayer online games and several works have focused on analyzing such e-sports (Chen et al. 2009).

To the best of our knowledge none of the existing works in the area of sport analytics addresses the problem setting or applications that we have considered in this paper. For a more in-depth coverage of the research in sports analytics we refer the reader to the book of Miller (2015).

4 Algorithms

The TEMP- NET- SUMM problem consists of finding both the k segments (sessions) and the h network summaries (modes) such that the score is minimized.

Given any set network of summaries, we may find the segmentation that minimizes the score function for that set, by using *dynamic programming*.

Proposition 4 *Consider a temporal network H, parameters k, h and R, and assume that the set of network summaries $\mathcal{C} = \{C_1, \ldots, C_h\}$ is given as input. The segmentation $\mathcal{S}$ that achieves the optimal value $p(H, k, h)$ can be computed in time $\mathcal{O}(kh\ell R)$ by dynamic programming.*

The dynamic-programming algorithm, DP, is based on the following recursive formula, which assumes as input the set of network summaries $\mathcal{C} = \{C_1, \ldots, C_h\}$.

$$o(i, k) = \min_{1 \le j < i} \{o(j, k-1) + \min_x d(\pi(H[j+1, i]), C_x)\},$$

where $o(i, k)$ denotes the cost of segmenting the time-interval $[1, i]$ in k segments.

To use this equality efficiently, we need to be able to compute the distance $d(\pi(H[j+1, i]), C_x)$ in constant time. Luckily, we can do this by maintaining two binary arrays for each centroid $x = 1, \ldots, h$, indicating if there is a particular edge in $\pi(H[j, i]) \cap C_x$ and $C_x \backslash \pi(H[j, i])$, respectively. These arrays are at most of length $\mathcal{O}(m)$. We compute the above recursive formula by starting with $j = i - 1$ and decreasing the value of j. Each time we decrease j, we need to add to the arrays the edges of time point j. This can be done in constant time. Due to the limitation of the length R of the segment, this leads to $\mathcal{O}(kh\ell R)$ running time.

A second technique to improve the speed of the method is a run-time optimization. We observe that between consecutive time points (i.e., consecutive iterations of j) if the edges at the new timepoint have all been seen before (if all the edges at timepoint j exist in $\pi(H[j+1, i])$, then $\pi(H[j, i]) = \pi(H[j+1, i])$), we may avoid recomputing the distances. Thus, for each $o(i, k)$, the distances will be computed in at most m cases. We expect the first optimization to provide a greater benefit for large values of m, while the second for cases where m is small relative to n.

On the other hand, if a segmentation $\mathcal{S}$ of the temporal network is given, the problem of finding the optimal set of h network summaries for that segmentation, reduces to the h-median problem in high-dimensional binary vectors, and it is computationally hard.

Proposition 5 *Given a k-segmentation $\mathcal{S}$ of a temporal network H, the problem of finding the optimal set of h network summaries is NP-hard.*

Proof The problem is known as the k-median clustering problem, which is NP-hard in the general case. The problem remains NP-hard when the points are restricted to vertices of the hypercube, known as the HYPERCUBE- SEGMENTATION problem (Kleinberg et al. 1998). □

Although the problem is NP-hard, one can use a standard clustering algorithm, such k-median, or k-means, to obtain a solution that will be good in most practical situations.

4.1 An iterative algorithm

From our discussion so far, it follows that given the network summaries we can find the optimal segmentation (for those summaries) by dynamic programming. Additionally,

Algorithm 1: Iterative algorithm

Data: H: the temporal network; k: number of segments; h: number of summaries; R: interval length bound.

```
C = random_summaries(H);
repeat
    S = segment(H, C, k, R);
    C = cluster(S, h);
until convergence;
```

given a k-segmentation of the temporal network, we can find a good set of summaries, via a clustering algorithm. This leads to an iterative algorithm that alternates between solving these two problems; depicted in Algorithm 1.

The Iterative algorithm starts off by creating random summaries; for each summary, each edge is added with probability 0.5. For the clustering phase, we have implemented the analogous of k-means++ (Arthur and Vassilvitskii 2007).

4.2 A greedy approach

In this section we describe two variants of a greedy approach. We start our discussion by considering a different interpretation of the TEMP- NET- SUMM problem: given the set of all edges, the end goal is to derive a k-segmentation of the network's time-span, so that the resulting segments are, when divided into h groups, as similar as possible to each other. Hence, our goal should be try to create similar segments. We follow a bottom-up approach, where initially each time point is a segment; hence we have n segments. The problem reduces to performing $n-k$ merges of consecutive segments, in order to finally acquire k segments. The merges should happen in a way that builds similar patterns among different segments.

The first approach relies on attempting to minimise the final score by greedily merging the most similar pairs of consecutive segments. At first, we need to compute the $n-1$ distances between timepoints t_i and t_{i+1}, for $i = 1, \ldots, n-1$. This can be done in $\mathcal{O}(\ell)$ time. We merge the pair of consecutive intervals that yield the lowest distance. Ties are broken arbitrarily. For the example of Fig. 1, at the first step, the algorithm would merge the single time-point segments of t_1 and t_2, or those of t_5 and t_6. In both cases, the segments' distance is 1.

After each merge, we need only to compute the distance between the newly formed segments and its two neighboring segments, left and right; the rest of the distances remain the same. In addition, we incorporate the limit on the length of the intervals by checking the length of the intervals before considering to merge them. We will do at most $n-k$ merges, and a single merge requires $\mathcal{O}(m)$ time. This yields a total running cost $\mathcal{O}((n-k)m)$. We refer to this method as the *simple greedy* method.

The above method favors the merge of similar consecutive segments. As a result, after each step, we can expect that the remaining segments would be increasingly different to each other. The original goal was the opposite. Hence, we may derive an additional variation of this algorithm. Instead of merging the most similar consecutive segments, at each step the algorithm merges those consecutive segments that

Algorithm 2: Simple Greedy Algorithm

Data: H: the temporal network; k: number of segments; h: number of summaries; R: interval length bound.

```
for i = 1, ..., n − 1 do
    compute d(intervals[i], intervals[i + 1]);
while |intervals| > k do
    i* = argmin d(intervals[i], intervals[i + 1]) ;
    merge(intervals[i*], intervals[i* + 1]);
    compute d(intervals[i*], intervals[i* + 1]);
    compute d(intervals[i* − 1], intervals[i*]);
```

when merged will result in being the most similar to any other existing segment. This approach is an order of magnitude slower than the simpler greedy approach; without using sophisticated data structures for keeping scores, the running time is $\mathcal{O}(n^3 m)$.

For the example of Fig. 1, at the first step, this approach would merge the single time-point segments of t_1 with t_2 since the result would be identical to that of t_4, or those of t_5 and t_6 to match that of t_3. Unlike the simple greedy heuristic, the second variant is able to find the optimal solution for the example of Fig. 1.

4.3 A factor-2 approximation algorithm

Finally we present a factor-2 approximation algorithm that is only useful for a limited set of cases. It relies on the idea that for the clustering problem, in any cluster there exists (at least) one data-point that minimises the sum of distances to the remaining data-points of the cluster. This is not the mean/median of the cluster in the optimal solution, but it can be used as an approximate solution to it. In our problem setting, as described previously, the set of possible points for the clustering instance consists of the set of graphs that correspond to all possible segments of the temporal network.

It follows that for finding the best network summaries $\mathcal{C} = \{C_1, \ldots, C_h\}$ we may use, as an approximate solution, graphs that correspond to existing segments. In particular, for each cluster, there exists one segment whose corresponding graph (its projection) minimises the sum of distances from all other segments in the cluster. Choosing that graph provides a factor-2 approximation for the cluster. And if we can find the approximate network summaries for all clusters, we can then apply the dynamic-programming algorithm to find the final segmentation, which will also be within a factor-2 approximation of the optimal solution.

As described previously, the candidate summaries are the projections of all possible segments of the temporal network. There are $\binom{n}{2h}$ such segments. In other words, we need to consider all possible h-size subsets of the set of all segments of the temporal network, then for each subset apply the DP algorithm, and keep the solution that yields the best score.

For $h = 1$, we need to find the single segment of the temporal network, which, when used as summary, yields the lowest score. For $h = 2$, we need to find the best pair of segments, and so on. It follows that the overall complexity of the method is $\mathcal{O}(n^{2h})$ multiplied by the complexity of the DP algorithm. Since h is in the exponent,

we expect this method to be highly inefficient for large datasets and for cases where $h \geq 2$.

5 Speeding up the dynamic program

We saw that if we know the centroid graphs we can compute the optimal segmentation with a dynamic-programming approach. Unfortunately, this method becomes too slow for very large sequences as R, the window size, must grow at least linearly with the sequence length. In this section, we present an algorithm that runs in quasi-linear time and provides an $(1+\epsilon)$ approximation. For notational simplicity, we ignore the window size constraint as it is trivial to incorporate.

5.1 Definition of auxiliary problem

Assume that we are given a temporal network H. Let us define

$$\text{in}(a, b, C) = |E(C) \backslash E(\pi(H[a, b]))|$$

and

$$\text{out}(a, b, C) = |E(\pi(H[a, b])) \backslash E(C)|.$$

Note that the symmetric distance $d(\pi(H[a, b]), C)$ used in the penalty function in Equation (1) is equal to $\text{in}(a, b, C) + \text{out}(a, b, C)$.

To solve the original problem faster, we consider another, more difficult, problem.

Problem 2 Given a temporal network H, a set of h summaries $\mathcal{C}$, a budget on the number of segments k, and a budget for edges o, find a segmentation $\left\{(a_j, b_j, C_j)\right\}_{j=1}^{k}$, minimizing

$$\sum_{j=1}^{k} \text{in}\big(a_j, b_j, C_j\big) \quad \text{such that} \quad \sum_{j=1}^{k} \text{out}\big(a_j, b_j, C_j\big) \leq o,$$

and $C_j \in \mathcal{C}$. We will denote the objective score of this segmentation by $q(H, k, o)$. We will abbreviate $q(H[1, i], k, o)$ by $q(i, k, o)$.

We can solve this problem exactly with the following recursive equality,

$$q(i, k, o) = \min_{C \in \mathcal{C}} \min_{o' \leq o} \min_{j \leq i} q\big(j-1, k-1, o'\big) + \text{in}(j, i, C), \tag{2}$$

such that $\text{out}(j, i, C) \leq o - o'$.

Using this equality directly is highly inefficient. Note that iterating over j requires $\mathcal{O}(n\ell)$ steps, alone, where n the number of time stamps and ℓ is the total number of timestamped edges. Our first step is to reduce this to $\mathcal{O}(\ell \log n)$ steps.

5.2 Optimizing the convex problem

We will now show that we can compute $\min_{j \le i} q\big(j-1, k-1, o'\big) + \text{in}(j, i, C)$ fast. We argue that this minimization problem is an instance of the following minimization problem.

Problem 3 Let n be an integer, and define $P = \{(x, y), 1 \le x \le y \le n\}$. Let $f : P \to \mathbb{R}$ be a function such that

$$f(x_1, y_1) - f(x_2, y_1) \le f(x_1, y_2) - f(x_2, y_2),$$

for $x_1 \le x_2 \le y_1 \le y_2$ Also let $g : P \to \{0, 1\}$ be a function such that $g(x_1, y_1) \le g(x_2, y_2)$ for $x_1 \le x_2 \le y_2 \le y_1$. Compute

$$b(i) = \arg\min_j \{f(j, i) \mid g(j, i) = 1\},$$

for every i. The ties are solved by picking the larger index.

It is easy to see that $f(j, i) = q\big(j-1, k-1, o'\big) + \text{in}(j, i, C)$ satisfies the conditions for f, and the function $g(j, i) = I[\text{out}(j, i, C) \le o - o']$, where $I[S] = 1$, if statement S is true, and 0 otherwise, satisfies the condition for g.

Let us now show the benefits of this setup. If f and g can be computed in constant time, then we can use SMAWK algorithm to compute b in $\mathcal{O}(n)$ time (Aggarwal et al. 1987). This is however not possible since we cannot compute f and g in constant time. Fortunately, we can obtain a solver with slightly worse running time. The key lemma for this solver states that b is monotonic.

Lemma 1 *Let f, g and b as stated in Problem 3. Then $b(i) \le b(j)$ for $i < j$.*

Proof Fix $i < j$ and assume that $b(i) > b(j)$. By definition of f,

$$f(b(j), i) - f(b(i), i) \le f(b(j), j) - f(b(i), j).$$

Due to our assumption, $g(b(j), i) \ge g(b(j), j) = 1$, and so the optimality of $b(i)$ guarantees that

$$f(b(i), i) \le f(b(j), i).$$

Combining the two inequalities leads to

$$0 \le f(b(j), i) - f(b(i), i) \le f(b(j), j) - f(b(i), j)$$

or $f(b(i), j) \le f(b(j), j)$, which contradicts the definition of $b(j)$. □

The trick is now to compute $b(i)$ in interleaved fashion, and use the previously computed values to bound possible candidates for $b(i)$. The order of solving $b(i)$ is as follows: We first split the data in half, say at c and compute the entry in $b(c)$. Lemma 1

Algorithm 3: ConOpt(f, g) solves Problem 3

```
1  n ← number of points;
2  s[i] ← −∞ for i = 1, ..., n;
3  b[i] ← 1 for i = 1, ..., n;
4  t ← ⌊log2(n − 1)⌋;
5  foreach Δ = 2^t, 2^{t−1}, ..., 1 do
6      i ← 1 + Δ;
7      while i ≤ n do
8          u ← b[i − Δ];
9          v ← i;
10         if i + Δ ≤ n then v ← b[i + Δ] foreach j = u, ..., v do
11             if f(j, i) ≤ s[i] and g(j, i) = 1 then
12                 b[i] ← j; s[i] ← f(j, i);
13         i ← i + 2Δ;
14 return b;
```

now implies that $b(i)$ in the first half should be smaller or equal than $b(c)$ and $b(u)$ in the second half should be larger or equal than $b(c)$. We continue recursively this divide-and-conquer approach. The pseudo-code is given in Algorithm 3. We have the following key result.

Proposition 6 *Let n, f and g as given in Problem 3. Assume that there is a data structure $D(j,i)$ that allows us to compute $f(j,i)$ and $g(j,i)$ in constant time. Moreover, assume that we can obtain $D(1,1)$ in $\mathcal{O}(n)$ time and we can update $D(j,i)$ to $D(j+1,i)$ or $D(j,i+1)$ in amortized $\mathcal{O}(\ell/n)$ time. Then* ConOpt *solves Problem 3 in $\mathcal{O}(\ell \log n)$ time.*

By amortized time, we mean that as we move indices i and j from 1 to n, the total update time is at most $\mathcal{O}(\ell)$.

Proof Define b^* to be the solution to Problem 3. Let us first prove the correctness by induction over Δ: We claim that for fixed Δ, after the while-loop, $b[i] = b^*(i)$ is the solution for any $i = 1 + (2k+1)\Delta \leq n$, where $k \geq 0$. If $\Delta = 2^t$. Then the only possible i is 1, so the claim holds automatically. Assume that the claim holds for 2Δ. Then $b[i-\Delta] = b^*(i-\Delta)$ and, if exists, $b[i+\Delta] = b^*(i+\Delta)$. Lemma 1 guarantees that $b[i-\Delta] \leq b^*(i)$, and $b^*(i) \leq b[i+\Delta]$ or $b^*(i) \leq i$. This is exactly the interval over which j iterates, which proves the claim.

To prove the running time first notice that we have at most $\mathcal{O}(\log n)$ different values of Δ. In addition, for a fixed Δ, i only increases during the iteration, and since u for a current i is equal to v for the previous i, j only increases for a fixed Δ. We can now use the data structure D by first initializing it to $D(1,1)$ before the while loop, and updating it whenever i or j increases. Since $i, j \leq n$, we use at most $\mathcal{O}(\ell)$ updates for a fixed value of Δ. This proves the proposition. □

To complete the argument, we need to show that we can compute $\text{in}(j,i)$ and $\text{out}(j,i)$ in amortized $\mathcal{O}(\ell/n)$ time. Note that i and j only move to the right. First, we can safely assume that each edge is represented by an integer between 1 and $m \leq n$.

Algorithm 4: FastDP$(H, \mathcal{C}, k, \epsilon)$, solves approximately the segmentation given the centroids.

```
ℓ ← number of edges in H;
n ← number of time points in H;
O ← ∅; o ← 0;
while o ≤ ℓk do
    append O with o;
    o ← ⌊1 + (1 + ϵ)o⌋;
s ← array of size n × k × (|O| − 1);
s[i, 1, x] ← min_{C∈𝒞} {in(1, i, C) | out(1, i, C) < (1 + ϵ)o_x};
foreach d = 2, ..., k do
    foreach C ∈ 𝒞 do
        foreach 1 ≤ x ≤ y < |O| do
            f(i, j) ← s[j − 1, d − 1, x] + in(i, j; C);
            g(i, j) ← out(i, j; C) ≤ (1 + ϵ)o_y − o_x;
            b ← ConOpt(f, g);
            update s[i, d, y] using b[i], if better;
return the segmentation responsible for s[n, k, x] with the lowest penalty p;
```

We use two arrays, one for in and the other for out, both of length m to store the counts of edges in a current interval. Everytime we move i or j we update the appropriate array, as well as the count of non-zero entries. Every edge is added or deleted only once, which gives us total update time of $\mathcal{O}(\ell)$. This will provide us with the needed data structure.

5.3 Approximation algorithm

We can now use the described technique to optimize Equation (2) over j and i but we still need to address o and o'. Note that they are upper-bounded by nk, which can be significantly large. The trick is not to use every possible value of o. Instead, it is enough to consider only a geometric sequence 1, $(1 + \epsilon)$, $(1 + \epsilon)^2$, and so on. To compensate for the ignored values, we need to relax the constraint in Equation (2) to $\text{out}(j, i, C) \leq (1 + \epsilon)o - o'$. That is, we allow the number of out-edges to be slighly higher. This leads to Algorithm 4.

Proposition 7 *The computational complexity of* FastDP$(\mathcal{S}, \mathcal{C}, k, \epsilon)$ *is* $\mathcal{O}(k|\mathcal{C}|\epsilon^{-2}\ell \log n(\log^2 \ell k))$.

Proof Let O as defined by FastDP. We must have $|O| \in \mathcal{O}(\log(\ell k)/\log(1 + \epsilon)) \subseteq \mathcal{O}(\log(\ell k)\epsilon^{-1})$. The number of ConOpt calls is $\mathcal{O}(|O|^2 k|\mathcal{C}|)$. The proposition now follows from Proposition 6. □

Our next step is to prove the approximation guarantee. In order to do that we need to show that the total number of out-edges of the obtained solution is not far from the exact solution.

Proposition 8 *Fix i, d, and y, and let $\mathcal{S}$ be the segmentation responsible for the score $s[i, d, y]$ by the end of* FastDP. *Let O as defined by* FastDP. *Let o_y be the yth entry in O. Let o' be the total number of out-edges in $\mathcal{S}$. Then $o' \leq (1 + d\epsilon)o_y$, and $s[i, d, y] \leq q(i, d, o)$, for any $o \leq (1 + \epsilon)o_y$.*

Proof We will prove this using induction over d. The result holds automatically for $d = 1$.

To prove the general case, let us assume that we have just computed b that is responsible for $\mathcal{S}$. Since f and g satisfy the conditions of Problem 3, the obtained b solves the problem $\min f(j, i)$, such that $g(j, i) = 1$. Note that g imposes weaker constraints than the enforced constraint in Eq. 2. This, and the induction assumption, implies immediately the second inequality.

To prove the first inequality, let us write r to be the total number of out-edges of $\mathcal{S}$, without the last segment, $r = o' - \text{out}(b[i], i)$. Induction assumption now states that $r \leq (1 + (d - 1)\epsilon)o_x$. We have

$$\begin{aligned} o' &= \text{out}(b[i], i) + r \\ &\leq (1 + (d - 1)\epsilon)o_x + (1 + \epsilon)o_y - o_x \\ &= (d - 1)\epsilon o_x + (1 + \epsilon)o_y \\ &\leq (1 + d\epsilon)o_y, \end{aligned}$$

which completes the proof. □

We can now state the main result of this section.

Proposition 9 FastDP$(H, \mathcal{C}, k, \epsilon/k)$ *returns a segmentation yielding $(1 + \epsilon)$-approximation guarantee in $\mathcal{O}(k^3|\mathcal{C}|\epsilon^{-2}\ell \log n(\log^2 \ell k))$ time.*

Proof The computational complexity follows immediately from Proposition 7. Let $\mathcal{S}^*$ be the segmentation optimizing p, and let o be the number of out-edges in $\mathcal{S}^*$. Let i be the number of in-edges in $\mathcal{S}^*$, $i + o = p(H, \mathcal{S}^*)$. Let $o_x \in L$ be the value such that $o_x \leq o \leq (1 + \epsilon)o_x$. Let $\mathcal{S}$ be the segmentation responsible for $s[n, k, o_x]$. Let o' be the number of out-edges in $\mathcal{S}$. Proposition 8 implies that

$$\begin{aligned} p(H, \mathcal{S}) &= s[n, k, o_x] + o' \\ &\leq s[n, k, o_x] + (1 + \epsilon)o_x \\ &\leq s[n, k, o_x] + (1 + \epsilon)o \\ &\leq i + (1 + \epsilon)o \\ &\leq (1 + \epsilon)p\big(H, \mathcal{S}^*\big). \end{aligned}$$

This completes the proof. □

Table 1 Datasets overview

Dataset	Time span n	Graph size m
NBA	11 315	147
Handball	38	21
Premier league	38	200
Twitter WorldCup	39 988	139 254
Enron	200 704	143 695
Reality mining	307	13 786

6 Experimental evaluation

In this section we present our empirical evaluation. Section 6.1 contains a description of the datasets, while in Sect. 6.2 we evaluate our algorithms. Finally, Sect. 6.3 contains our case-studies on real datasets.

6.1 Datasets

For our experimental evaluation we used the following dynamic networks. Table 1 contains an overview of the datasets.

NBA Dallas Mavericks This is a dataset with play-by-play information of the National Basketball Association's matches for the season 2014–2015. Two players interact if they are present on the court at the same time. Substitutions are allowed only during possession-changes, so the time granularity is on a per-possession basis. We focus on the Dallas Mavericks team.

Handball (Pers et al. 2006) the CVBase06 handball dataset. Vertices correspond to players and edges between players to passes of the ball. It contains 38 different plays (series of passes) that lead to a shot, a steal or any other action that interrupts the passes among the team's members.

Premier League This dataset contains the starting lineups of every football (soccer) match of the Premier League (English national championship) during the season 1996-'97. Two players interact if they appear on the same team's lineup of a given match. We focus on the Arsenal FC team.

Twitter World Cup (Rayana and Akoglu 2016) This is a collection of tweets from before and during the 2014 football World Cup. The vertices are hashtags or users and an interaction is a co-occurrence in a tweet. We set the time granularity to one hour.

Enron dataset (Klimt and Yang 2004) This is the collection of email messages exchanged between the employees of Enron. In this network, each employee is a vertex and an email between two employees denotes an interaction.

Reality Mining (Eagle and Pentland 2006) This is a real-life social network. Each vertex corresponds to a human. An interaction takes place if two humans are in proximity,

Table 2 Indicative running times & scores for different parameter values

Dataset	(k, h)	Score		Runtime (s)	
		Iterative	Greedy	Iterative	Greedy
Twitter	(10, 5)	23	109,567	173,902	234
Twitter	(20, 5)	62,061	166,710	199,409	200
Enron	(2, 1)	116,894	121,294	282,346	9114
RealityM.	(10, 1)	21,185	31,424	124	2
RealityM.	(10, 5)	488	16v025	291	2

and the mobile phone's Bluetooth component belonging to one human discovers the phone of the other.

Finally, to test the behavior of our methods with respect to specific parameters, we create artificial dynamic networks. For each time point, we create an instance of $G(n, p)$ Erdős-Rényi graphs.

In order to enable repeatability of our experiments, and to facilitate future research efforts, we provide our methods' source code[1] and the NBA, Handball, and Premier League datasets[2].

For the Enron dataset and the Twitter World Cup datasets, we have removed any edges that appear only once. The rationale is that since we are interested in finding recurring patterns, we know in advance that such edges will only add to the score (for any $k > 1$). Due to the datasets' size these edges that appear only once will skew the obtained scores.

6.2 Methods benchmark

In this section we provide the results from benchmarking our methods. The goal is to gain insights into their performance in terms of quality of solutions and running times.

Due to the fact that some of the methods are randomized, unless stated otherwise, we executed each of those methods 3 times. We report the best score of the 3 executions. The reported runtimes are the mean runtimes. In addition, we have limited the iterative algorithm to 5 iterations between the segmentation and the clustering phase.

Table 2 reports runtimes and scores over some of the real datasets, for different values of k and h. We omit the results for the sophisticated greedy and the 2-approximation algorithm, since their runtime for the larger datasets exceeded our 5-day margin. However, the important insight is that the sophisticated greedy does not offer any advantage over the simpler variant. In particular, its score was worse or similar to that of the simple and its running time several orders of magnitude longer. Between Iterative and Simple Greedy, we notice a clear trade-off between performance and running time. Iterative provides better results but requires more time.

[1] https://doi.org/10.5281/zenodo.290629

[2] https://doi.org/10.5281/zenodo.160509

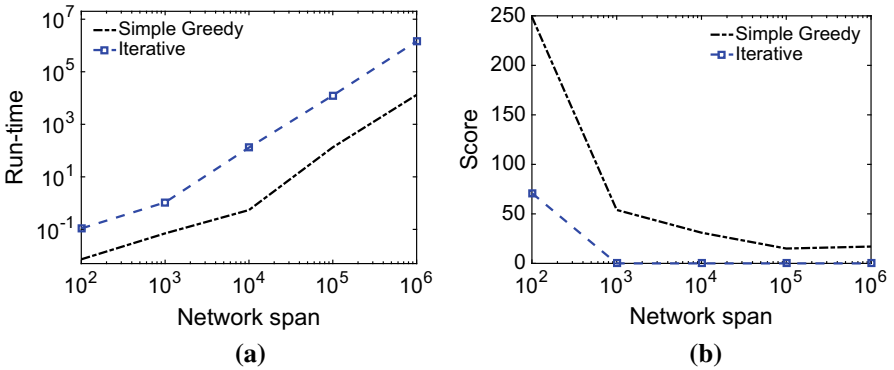

Fig. 3 Benchmark for varying time span. **a** Running times (log-log). **b** Scores

Table 3 Scores for the experiments with synthesized ground truth, for different values of segments k, summaries h, network time-span n, and number of distinct edges m

Parameters				Scores		
k	h	n	m	Iterative	Simple Greedy	Soph.Greedy
10	2	1000	50	0	14	43
20	2	1000	50	0	49	118
10	2	10,000	50	0	23	68

We benchmark our methods in terms of run-time, over different values of time span n, graph size m, and parameters k and h, using Erdős-Rényi graphs; the value of only one parameter is changed every time. Figure 3 depicts the results for different values of n. When varying k and h, the Iterative appears to perform faster as h increases, but without a solid trend. We also observe that all methods scale linearly with respect to m.

We examine the quality of the solutions achieved by our methods, with "planted" (pre-defined) sessions and modes. The idea is that we already know the optimal solution and its value. First, we create a set of h distinct random sub-networks of equal length. Then we synthesize the temporal network as a sequence of k random selections (with replacement) from the h sub-networks. Clearly, for those values of k and h, the optimal solution yields zero score; $p(H, k, h) = 0$.

Table 3 depicts the mean scores for different values of k and h, after 10 independent executions. We remind that maximum possible score in any instance of the problem is km. Iterative is able to retrieve the optimal solution in all cases. The greedy approaches never manage to achieve that; again, the sophisticated greedy is worse.

6.2.1 Combining the iterative and greedy methods

Our iterative implementation begins of with a set of random summaries and then alternates between segmenting the timeline and clustering the segments. Instead, the simple greedy algorithm is the fastest and returns a segmentation of the timeline based on the actual timeline data. Due to this, we investigate whether starting the iterative

Table 4 Scores when benchmarking Iterative versus Greedy+Iterative

Dataset	(k, h)	Iterative	Greedy+Iterative	Greedy
RealityM.	(10, 1)	13,500	14,176	22,832
RealityM.	(10, 3)	28,127	30,607	32,202
NBA	(10, 1)	207	207	341.4
NBA	(200, 4)	2547.9	2585.3	3622.5

algorithm from the segmentation of the simple greedy approach provides any benefit. Table 4 depicts the mean scores from 10 independent comparisons (best of 1). We have added the mean scores of the stand-alone greedy for reference.

We notice that if Iterative starts with the segmentation provided by the greedy algorithm, the results are in the general case worse than executing Iterative with random summaries; only for the NBA dataset and $k = 10$, $h = 1$ did both methods reach the same solution. We presume that the iterative procedure gets trapped in a local optimum. On the other hand, starting from the segmentation provided by the greedy method, allows the iterative procedure to reach a solution faster. The average running time was reduced by 14 and 19% for the Reality Mining and NBA datasets respectively. Hence, there is a trade-off between performance and running time.

6.3 Mining social networks

In this section we present our study of mining real temporal networks taken from the field of sports and social networks. We demonstrate the applicability of our methods on datasets related to basketball, football (soccer) and handball. In particular we summarize: (i) basketball in-game team formations over a season, (ii) handball in-game passing behavior (iii) football team lineups over the whole season. In addition, we explore the Twitter World Cup dataset.

The goal of the following experiments is not only to demonstrate the broad applicability of our methods, but additionally to prove their effectiveness and usefulness for the specific field of Sports Analytics. We aim to simulate part of the analysis performed by professionals of the field by discovering meaningful modes and their corresponding segments in temporal networks of varying time-granularity taken from a range of sports. Finally, in Sect. 6.3.4, by analyzing the Twitter dataset, we make a first attempt to identify the significance and impact of various sports events on social media.

6.3.1 Summarizing basketball in-game team formations

We study the NBA dataset and attempt to gain helpful insights for the team of the Dallas Mavericks. The goal is to discover the common team formations. In other words, we want to identify in what combinations does the coach use his players.

In a professional setting, this would provide knowledge and insights on how to defend; in modern basketball especially, understanding the combination of skills in a

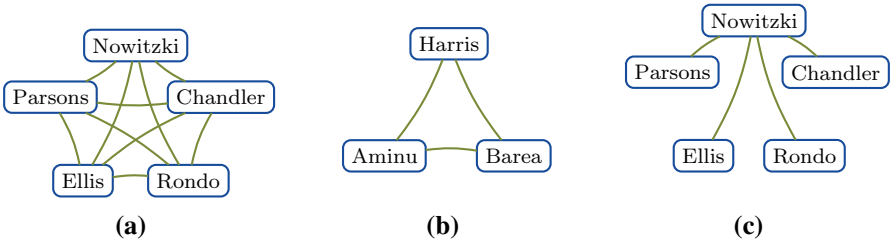

Fig. 4 Summaries discovered in the NBA dataset. **a** The smallest summary for $k = 200$, $h = 8$. **b** The intersection of the remaining summaries, for $k = 200$, $h = 8$. **c** The most common summary for other values of k, h

team is critical for deciding on the offensive play and, consequently for the opponents, how to defend (Obradovic 2007). This analysis is also useful for the sports of ice-hockey, to discover *lines*, and in gridiron football, or North American football, to identify frequent line-ups.

The dataset has information regarding 56 games, and since NBA basketball matches have 4 quarters but rarely players are on the pitch for a whole quarter, we set $k = 200$ and $h = 8$.

Using the iterative algorithm, the best solution we acquire has cost 2088 (for the simple greedy algorithm the score was 3187). The smallest summary in size is the 5-clique, presented in Fig. 4a. The remaining summaries are larger, and overlap significantly in pairs. The intersection of all the remaining summaries is a 3-clique presented in Fig. 4b, while in pairs the intersection may consist of up to 34 edges. The 5-clique also appears if we set $k = 200$ and $h = 4$.

By modifying the values of k and h, or when the iterative algorithm fails to find the 5-clique, the most frequent summary is a 5-vertex star-graph with the same vertices as the previous 5-clique; presented in Fig. 4c. The above results correlate with the club's payroll. The 5 players in the clique were the most highly paid in the roster. The other 3 players are substitutions players. In addition, the same order applies in total playing time for that season.

6.3.2 Summarizing in-game team-passing activity; handball

Our methods also apply to play-by-play game summarization. We apply our Iterative algorithm to the CVBase06 handball dataset. The goal is to summarise a team's strategy using as few modes as possible.

The dataset contains 38 different plays (series of passes) that lead to a shot, a steal or any other action that interrupts the passes among the team's members.

By examining different values for h, and setting $k = n = 38$, we discover that the number of unique plays is 26 (if we don't consider the counts of passes between each pair of players).

Indicatively, Fig. 5 depicts two of the three summaries discovered by using the Iterative algorithm with $k = 14$ and $h = 3$. The total cost ($p(H, 14, 3)$) is 12. The summary of Fig. 5a corresponds to 4 plays, inducing a cost of 6, while that of Fig. 5b corresponds to a unique play. The third summary contains a single pass between

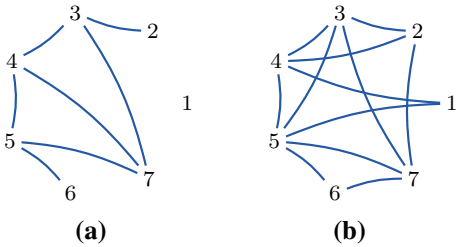

Fig. 5 Discovered summaries of the handball in-game passing plays, for $k = 14$ and $h = 3$. The third summary is a single edge between players '3' and '4'. **a** Mode 1. **b** Mode 2

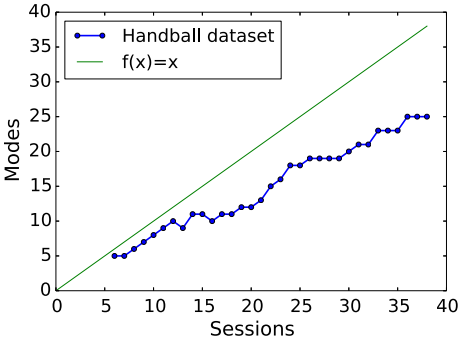

Fig. 6 Minimum number of modes (h, summaries) required to perfectly summarize the Handball dataset, as a function of the number of sessions (k, segments)

players '3' and '4', and is the centroid of the single- or two-pass plays, inducing the remaining cost.

Due to the fact that the players are only denoted by an id number, and hence the results are not informative, we plot the minimum number of summaries (h) required to perfectly summarize the Handball dataset as a function of the number of sessions (k); the results are depicted in Fig. 6. That is, the found summaries cover perfectly the found segments, $p(H, k, h) = 0$. For example, if we want to divide the team's activity into 25 segments, then we can perfectly summarize them using only 18 summary graphs.

Summarizing in-game team-passing activity can be used for both the opponents' and own team. For opponents, it provides direct insight on how they play, revealing patterns and roles, that can be beneficial for beating them. For coaching staff who analyze the plays of their own team, the above approach provides an evaluation measure for the team's performance. Consider for example a team that wants to play in a highly versatile way, then the minimum value of h that yields $p(H, k, h) = 0$ should be high. Alternatively, if the team's goal is to be very principled and not deviate from the coaches' instructions, then the score $p(H, k, h)$, with h equal to the number of agreed plays, provides a direct evaluation of the team; lower values indicate better adherence to the plan.

6.3.3 Identifying modes in football (soccer) lineups

We can apply our methods to summarising networks of more coarse-grained time granularity, such as football-team starting lineups. The question we answer is the

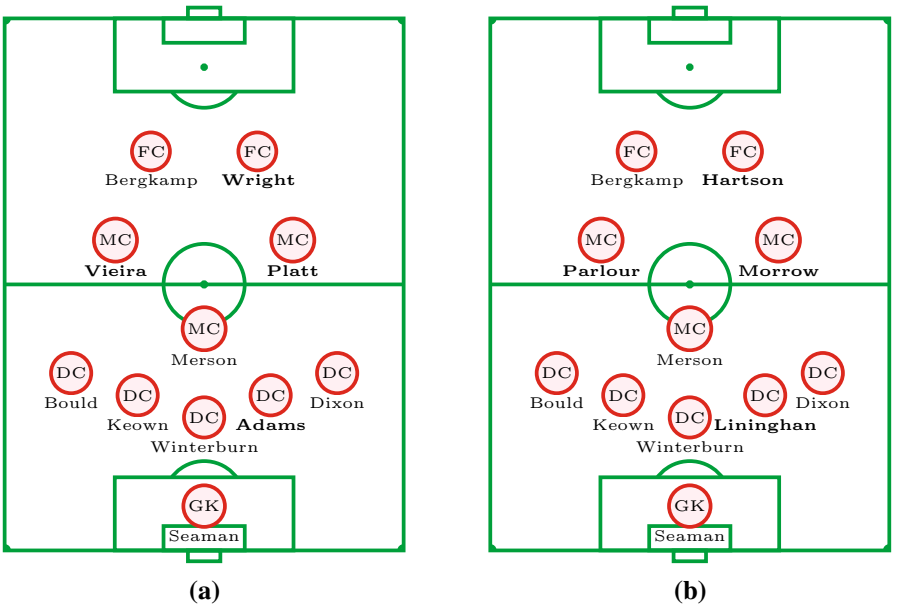

Fig. 7 Two of the three summaries discovered in Arsenal FC's starting lineups during the 1996–1997 Premier League season. These two summaries are 11-cliques, resulting in the above 11-player lineup summaries. The parameters were set to $k = 20$ and $h = 3$. **a** Regime 1. **b** Regime 2

following: "given one football team's starting lineups over the whole season, can we summarise them? In other words, can we identify which groups of players generate the lineups over the course of time?". We look into the staring lineups of Arsenal FC, during the 1996–1997 Premier League (national championship) competition.

We apply our Iterative method. By setting $k = n = 38$ and varying the number of modes h, we discover 25 unique lineups. The next question is whether we can summarize the evolution of the lineups with fewer sessions and modes. For $k = 20$ and $h = 3$, we obtain three summaries. Two of them are 11-cliques and correspond to 11-player starting lineups; they are depicted in Fig. 7. These two summaries are the centroids for 14 and 5 of the sessions respectively. The third summary (figure omitted) is not a clique and consists of 18 vertices (players). It is matched to only a single session.

For $k = 10$ and $h = 3$, one of the summaries is identical to that in Fig. 7a and it corresponds to 8 out of 10 sessions.

6.3.4 Analyzing Twitter activity during the 2014 World Cup

We further demonstrate the applicability of our methods beyond temporal networks corresponding to teams. We focus on online social media activity, and in particular we analyze the Twitter World-Cup dataset. Tweets (twitter messages) are aggregated into 24 hour granularity, resulting in 79 days. The network's nodes are individual

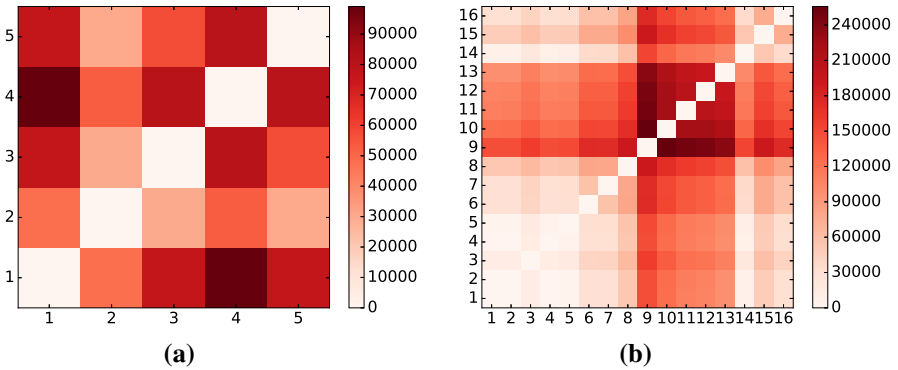

Fig. 8 Heatmaps based on the symmetric difference of the edges, for the Twitter World Cup dataset, with $k = 16$ and $h = 5$. Sessions are ordered by time. **a** h modes' heatmap. **b** k sessions' heatmap

hashtags (for example #fifa), and an edge exists between two hashtags if they co-occur in a tweet. We aim to gain insights into the evolution of tweets' content over time.

We apply the Iterative algorithm by setting $k = 16$ ($79/16 \approx 5$ days), $h = 5$, and $w = 1.5$. Figure 8a depicts the heatmap of the distance matrix between the summaries; darker corresponds to greater distance. The output summaries are singificantly different from each other.

Figure 8b depicts the heatmap of the distance matrix between segment-graphs. We notice that the first 5 segment graphs, together with the 13th have relatively low distance. This is due to the fact that the graphs are small. While from the 8th to the 12th are large graphs that are significantly different from each other. The mapping of segments to cores is: 2, 2, 2, 2, 2, 3, 5, 4, 2, 2, 2, 2, 2, 2, 1, 2, respectively. While we don't discover any session periodicity in the tweets, we discover that overall the content evolves over time.

We also discover that there is an overlap among several modes. For example, the third, fourth and fifth summaaries share 598 edges. This is expected since hashtags such as *#sports #football*, *#soccer*, *#brazil* and others correspond to very generic but yet relevant terms for that period. Moreover, these hashtags are high-degree nodes in the summary-graphs. On the other hand, by examining the session-graphs, we notice hashtags that are highly connected but their temporal locality is very narrow; for example *#brazilvsgermany* [3] co-appears with 914 other hashtags but only within a single segment. In summary, we witness two phenomena: first, the content changes significantly over time but new terms manage to become very central. Second, there is a body of terms that remain relevant throughout many sessions.

Figure 9 depicts the aforementioned discovered modes; for visual clarity, the graphs contain only those nodes with degree greater than the 50th largest degree value. We

[3] This hashtag refers to the first semi-final of the 2014 World Cup held in Brazil. Germany beat home-team Brazil by 7–1.

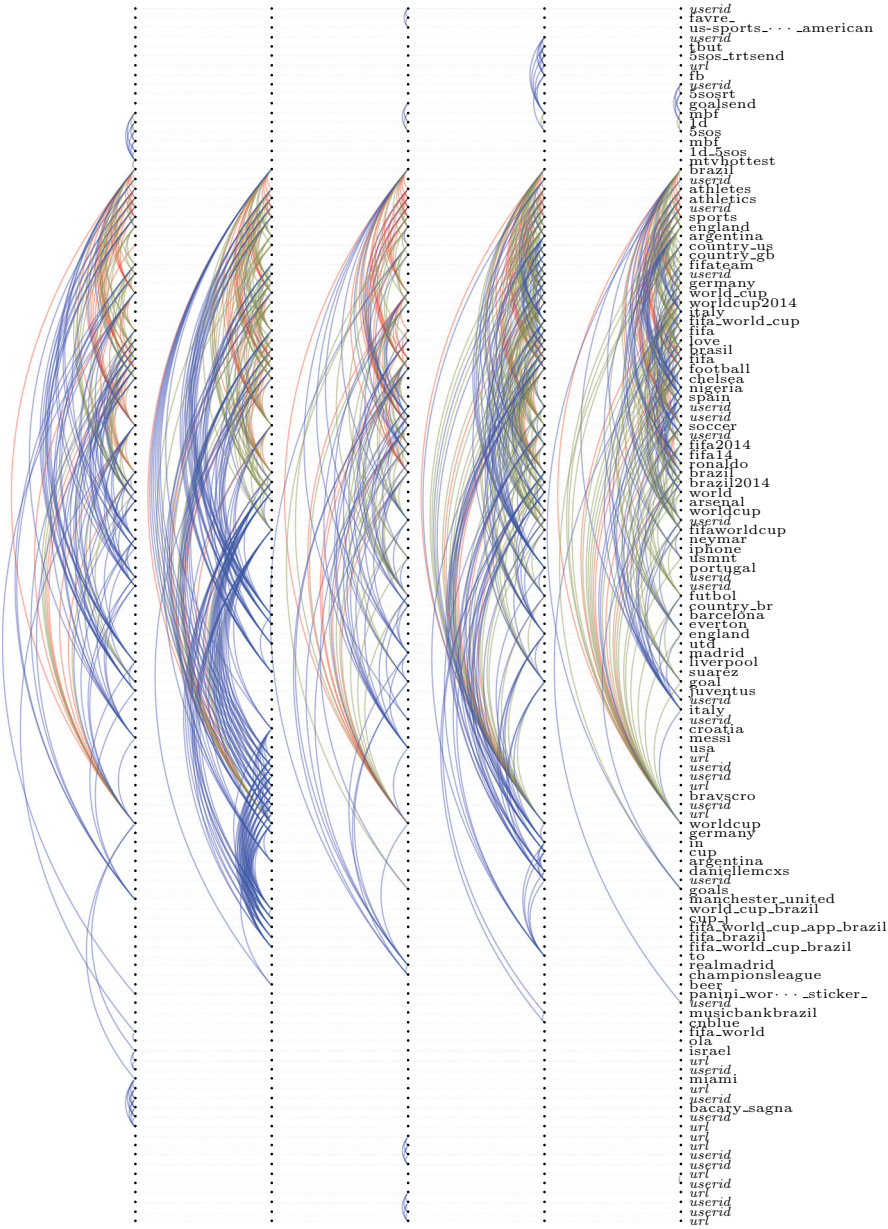

Fig. 9 Linear layouts of 5 summaries discovered for the Twitter World Cup dataset, for $k = 38$ and $h = 2$. The plotted graphs contain only those vertices with degree greater or equal to the 50th largest degree value of each graph. *Blue edges* are unique to a summary, *red edges* occur in each summary, and the remaining edges are *colored green* (Color figure online)

notice that the hashtags in the sumamries form communities. For example, those for football (soccer) are separated from those of American football. Similarly, there is a cluster of hashtags relating to music bands, that also happens to re-appear.

7 Concluding remarks

We considered the problem of summarizing temporal networks via discovering recurring activity. Our underlying assumption is that the temporal network operates in a small number of different modes, and the life-span of the network can be summarized by time-intervals in which the network operation makes transitions between these different modes. Among other applications, our problem formulation is motivated by discovering recurring team formations and recurring team strategies in sports analytics.

We formally defined the temporal-network summarization problem, and proved its NP-hardness. We then presented a number of different algorithms, including an iterative algorithm based on dynamic programming and clustering, an approximation algorithm that is practical only for one mode, as well as heuristic methods based on greedy approaches. Another technical contribution is a quasi-linear $(1+\epsilon)$-approximation segmentation algorithm, which is used in the inner loop of the iterative algorithm.

We demonstrated the practical qualities of our methods with an evaluation on real-world and synthetic datasets. Among the greedy approaches, the simple is faster than the sophisticated and in our experiments it still managed to achieve better scores. The iterative algorithm discovers solutions of better quality, compared to the simple greedy approach, but can be up to two orders of magnitude slower. Surprisingly, when setting as a starting point of the iterative algorithm the solution found by the simple greedy approach, compared to starting from a random state, the running time is reduced but solution quality deteriorates.

In addition, we performed several case-studies in which we use our approach to obtain meaningful insights. When analyzing the basketball dataset, our algorithms were able to discover results that correlate with other information such as player salary. In the future it would be interesting to investigate strategies for combining our methods together with other information such as score at each time point. This would facilitate coaches in getting an even better understanding of the teams under examination. For example, which team formation scores more points or which conceds the least. The same applies not only to the football starting lineups analysis but also to the lineups after any substitutions.

In the handball passing activity dataset, we examined the total number of modes under which a team operates, which are required to perfectly summarize the whole game. It would be interesting to see how this number correlates with the outcome of the game in various sports. Whether a low number of modes suggests that the team is able to perfectly execute the plays instructed by the coach, or whether a more diverse gameplay is the key to victory.

We also examined the applicability of our methods to Twitter. We discovered that certain hashtags have a short lifetime but become very popular since they co-occur with a high number of other hashtags. On the other hand, another group of hashtags remain prevalent throughout the whole network time span.

Appendix: Proof of NP-hardness

To prove the np-hardness we use the following problem.

Problem 4 (Satisfy) Assume that we are given q formulas over ℓ variables $\{v_i\}$ of form $\neg z = x \wedge y$, where x, y, and z are boolean variables or their negations. Decide whether these clauses can be simultaneously satisfied with v_1 being set to true.

Proposition 10 SATISFY *is NP-complete.*

Proof We will prove the hardness by reduction from 3SAT. Assume an instance of 3SAT with n variables and m clauses.

For each ith clause with two literals $x \vee y$, add $\neg c_i = \neg x \wedge \neg y$.

For each ith clause with three literals $x \vee y \vee z$, add two formulas $h_i = \neg x \wedge \neg y$ and $\neg c_i = h_i \wedge \neg z$.

If the ith clause contains one literal x, then refer to x as c_i.

Add $m-1$ variables $v_1, \ldots, v_{m-1}$, and formulas $v_i = v_{i+1} \wedge c_i$, for $i = 1, \ldots, m-2$, and $v_{m-1} = c_{m-1} \wedge c_m$.

It follows that ith clause can be satisfied if and only if c_i can be set to true. All c_is can be set to true if and only if v_1 can be set to true. □

Proposition 11 $(k, 1)$*-segmentation is NP-hard.*

Proof We will prove the hardness by reduction from SATISFY. Assume that we are given an instance of SATISFY with q formulas and ℓ variables.

We begin by specifying the vertices. The total number of vertices is $1+3+2\ell+r$, where $r = (20q + 12\ell + 2)(3 + 2\ell)$.

The first vertex is α, and every edge will be adjacent to α. The next three vertices are t_1, t_2, and t_3. Our construction will make sure that $(\alpha, t_i) \in E(G)$.

The next 2ℓ vertices correspond to the variables and their negations, we will denote them by v_i and u_i, for $i = 1, \ldots, \ell$. We will denote by X the set of possible edges between α and these vertices. Define $X' = X \setminus \{(\alpha, u_1), (\alpha, v_1)\}$.

Finally, the last r vertices are auxiliary vertices that will allow us to force segmentation borders. We will denote the set of possible edges between these vertices and α by B.

Our interation network consists of 3 parts, which in turn consists of sections. All these sections and parts are combined consecutively.

The first part, say P_1, consists of 2ℓ sections, each containing 5 time points. The first ℓ sections are defined as

$$\begin{array}{rl}
(\alpha, v_i): & 1\ 1\ \ \ \ \ \\
(\alpha, u_i): & \ \ \ \ 1\ 1 \\
(\alpha, t_1): & 1\ 1\ \ \ 1\ 1 \\
(\alpha, t_2): & 1\ 1\ \ \ 1\ 1 \\
(\alpha, t_3): & \ \ 1\ 1\ 1 \\
\text{for every } e \in B: & 1\ \ \ 1\ \ \ 1
\end{array}$$

They last ℓ sections are copies of the first ℓ sections, except that they also contain the remaining edges from X at 1st, 3rd, and 5th time point.

The second part, say P_2, consists of $2q$ sections, each containing 7 time points. Let $c_i = (\neg z = x \wedge y)$ be the ith formula. By using the same letters to represent the corresponding vertices, taking account negations, we define the ith section, where $i = 1, \ldots, k$, as

$$\begin{array}{rc}
(\alpha, x): & 1\ 1\ \ \ \ 1 \\
(\alpha, y): & 1\ \ \ 1\ 1 \\
(\alpha, z): & \ \ 1\ 1\ 1\ \ \ 1 \\
(\alpha, t_1): & 1\ 1\ \ 1\ 1\ 1\ 1 \\
(\alpha, t_2), (\alpha, t_3): & 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
\text{for every } e \in B: & 1\ \ 1\ \ 1\ 1\ 1
\end{array}$$

The $(q + i)$th section is a copy of ith segment, except that they also contain the remaining edges from X at 1st, 3rd, and 5th–7th time points.

The last part, say P_3, consists of $10q + 6\ell + 2$ sections, each consisting of 1 single time point. Each section contains B, (α, t_i), and (α, u_1). Moreover, every even section contains edges in X'.

We set $k = 20q + 12\ell + 2$. We claim that SATISFY is true if and only if the optimal segmentation has a score of

$$\sigma = |P_1|/2 \times (3(2\ell - 2) + 2) + |P_2|/2 \times (5(2\ell - 3) + 12) + |P_3|/2 \times (2\ell - 2) \quad .$$

We will prove this in several steps.

Step (i): *Every $e \in B$ is contained in every segment exactly once.* First, note that this segmentation is possible since B occurs at k different time points, the optimal cost of any such segmentation is bounded by $r/2$, the number of possible edges times half the number of segments. Note that each $e \in B$ occurs at the exact same time point. Thus there is an optimal solution with every $e \in B$ either present or absent from the core. Assume that there is a segment that disagrees with the core. Then the cost is at least r. Consequently, every segment must contain every $e \in B$. Since B occurs at k different time points, each segment can contain only one instance of each $e \in B$.

Step (ii): It follows immediately, that the borders of the sections are included in the borders of the optimal segmentation. Moreover, each section in P_1 part is divided into 3 segments, each section in P_2 is divided into 5 segments, each section in P_3 corresponds to exactly 1 segment.

Step (iii): $(\alpha, t_i) \in E(G)$, $u_1 \in E(G)$ *and* $v_1 \notin E(G)$. This follows immediately from the fact that each section in P_3 corresponds to one segment, and $|P_3| > k/2$, that is, P_3 contains the majority of the segments.

Step (iv): *The cost of ith and $(i + 1)$th section in P_1 is at least* $3(2\ell - 2) + 2$. *This bound is reached if and only if G contains either u_i or v_i, but not both.* First note, that the middle segment in both sectons contains the 3rd time point. This means that the remaining edges in X will occur exactly 3 times in 6 segments. Thus, they induce a cost of $3(2\ell - 2)$. A brute-force enumeration now implies that the involved edges induce a cost of at least 1, and this is possible if and only if G contains either u_i or v_i, but not both.

Step (v): *The cost of* i*th and* $(i+1)$*th section in* P_2 *is at least* $5(2\ell-3)+12$. *This bound is reached if and only if* $(\alpha, z) \notin E(G) \Leftrightarrow (\alpha, x) \in E(G)$ *and* $(\alpha, y) \in E(G)$. First note, that the 2nd segment in both sectons contains the 3rd time point, and the 4th and 5th segments consists of exactly one time point. This implies that the remaining edges in X will occur exactly 5 times in 10 segments. Thus, they induce a cost of $5(2\ell-3)$. A brute-force enumeration now implies that the involved edges induce a cost of at least 6, and this is possible if and only if $(\alpha, z) \notin E(G) \Leftrightarrow (\alpha, x) \in E(G)$ and $(\alpha, y) \in E(G)$.

Step (vi): *The cost of an odd and even section in* P_3 *is equal to* $2\ell-2$. This follows from the fact that the edges in X' occur exactly once in these two sections.

Step (vii): Steps *(iv)–(vi)* imply that σ is a lower bound for the optimal score. This bound is reached if and only if, the lower bounds of each section is reached. This can happen if and only if each sentence in SATISFY can be satisfied. □

References

Aggarwal A, Klawe M, Moran S, Shor P, Wilber R (1987) Geometric applications of a matrix-searching algorithm. Algorithmica 2(1–4):195–208

Alamar BC (2013) Sports analytics: a guide for coaches, managers, and other decision makers. Columbia University Press, New York

Appan P, Sundaram H, Tseng B (2006) Summarization and visualization of communication patterns in a large-scale social network. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, pp 371–379

Araujo M, Papadimitriou S, Günnemann S, Faloutsos C, Basu P, Swami A, Papalexakis EE, Koutra D (2014) Com2: fast automatic discovery of temporal (comet) communities. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, pp 271–283

Arthur D, Vassilvitskii S (2007) k-means++: the advantages of careful seeding. In: ACM-SIAM symposium on discrete algorithms, society for industrial and applied mathematics, pp 1027–1035

Asur S, Parthasarathy S, Ucar D (2009) An event-based framework for characterizing the evolutionary behavior of interaction graphs. ACM Trans Knowl Discov Data 3(4):16:1–16:36

Bellman R (1961) On the approximation of curves by line segments using dynamic programming. Commun ACM 4(6):284. doi:10.1145/366573.366611

Berlingerio M, Bonchi F, Bringmann B, Gionis A (2009) Mining graph evolution rules. In: European conference on machine learning and knowledge discovery in databases, pp 115–130

Chen KT, Jiang JW, Huang P, Chu HH, Lei CL, Chen WC (2009) Identifying mmorpg bots: a traffic analysis approach. EURASIP J Adv Signal Process 2009:3

Crowder M, Dixon M, Ledford A, Robinson M (2002) Dynamic modelling and prediction of english football league matches for betting. J R Stat Soc D 51(2):157–168

Denman H, Rea N, Kokaram A (2003) Content-based analysis for video from snooker broadcasts. Comput Vis Image Underst 92(23):176–195

Eagle N, Pentland A (2006) Reality mining: sensing complex social systems. Pers Ubiquit Comput 10(4):255–268

Eppstein D, Galil Z, Italiano GF (1998) Dynamic graph algorithms. CRC Press, Boca Raton

Gao X, Xiao B, Tao D, Li X (2010) A survey of graph edit distance. Pattern Anal Appl 13(1):113–129

Gift P, Rodenberg RM (2014) Napoleon complex: height bias among national basketball association referees. J Sports Econ 15(5):541–558

Gionis A, Mannila H (2003) Finding recurrent sources in sequences. In: International conference on research in computational molecular biology, RECOMB, pp 123–130

Goldsberry K (2012) Courtvision: new visual and spatial analytics for the nba. In: MIT sloan sports analytics conference

Greene D, Doyle D, Cunningham P (2010) Tracking the evolution of communities in dynamic social networks. In: IEEE of international conference on advances in social network analysis and mining, pp 176–183

Gudmundsson J, Horton M (2016) Spatio-temporal analysis of team sports—a survey. arXiv preprint arXiv:1602.06994
Guha S, Koudas N, Shim K (2006) Approximation and streaming algorithms for histogram construction problems. ACM Trans Database Syst 31(1):396–438
Halvorsen P, Sægrov S, Mortensen A, Kristensen DK, Eichhorn A, Stenhaug M, Dahl S, Stensland HK, Gaddam VR, Griwodz C, et al (2013) Bagadus: an integrated system for arena sports analytics: a soccer case study. In: Proceedings of the ACM multimedia systems conference. ACM, pp 48–59
Harville D (1980) Predictions for national football league games via linear-model methodology. J Am Stat Assoc 75(371):516–524
Hayet JB, Mathes T, Czyz J, Piater J, Verly J, Macq B (2005) A modular multi-camera framework for team sports tracking. In: IEEE conference on advanced video and signal based surveillance, pp 493–498
Heinen T (1996) Latent class and discrete latent trait models: similarities and differences. Sage Publications, Inc, Thousand Oaks
Henzinger M, King V (1999) Randomized fully dynamic graph algorithms with polylogarithmic time per operation. J ACM 46(4):502–516
Himberg J, Korpiaho K, Mannila H, Tikanmäki J, Toivonen H (2001) Time series segmentation for context recognition in mobile devices. In: IEEE international conference on data mining, pp 203–210
Holm J, De Lichtenberg K, Thorup M (2001) Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J ACM 48(4):723–760
Holme P, Saramäki J (2012) Temporal networks. Phys Rep 519(3):97–125
Hvattum LM, Arntzen H (2010) Using elo ratings for match result prediction in association football. Int J Forecast 26(3):460–470
Ide T, Kashima H (2004) Eigenspace-based anomaly detection in computer systems. In: ACM SIGKDD international conference on knowledge discovery and data mining
Kasiri-Bidhendi S, Fookes C, Morgan S, Martin DT, Sridharan S (2015) Combat sports analytics: boxing punch classification using overhead depthimagery. In: IEEE International Conference on image processing (ICIP), pp 4545–4549
Kleinberg J, Papadimitriou C, Raghavan P (1998) Segmentation problems. In: ACM symposium on theory of computing, pp 473–482
Klimt B, Yang Y (2004) The enron corpus: a new dataset for email classification research. In: Machine learning: ECML 2004. Springer, pp 217–226
Kostakis O (2014) Classy: fast clustering streams of call-graphs. Data Min Knowl Disc 28(5–6):1554–1585
Kumar R, Calders T, Gionis A, Tatti N (2015) Maintaining sliding-window neighborhood profiles in interaction networks. In: European conference on machine learning and knowledge discovery in databases. Springer, pp 719–735
Lucey P, Bialkowski A, Carr P, Morgan S, Matthews I, Sheikh Y (2013a) Representing and discovering adversarial team behaviors using player roles. In: IEEE conference on computer vision and pattern recognition, pp 2706–2713
Lucey P, Oliver D, Carr P, Roth J, Matthews I (2013b) Assessing team strategy using spatiotemporal data. In: ACM SIGKDD international conference on knowledge discovery and data mining, pp 1366–1374
Maheswaran R, Chang YH, Henehan A, Danesis S (2012) Deconstructing the rebound with optical tracking data. In: MIT sloan sports analytics conference
Miller TW (2015) Sports analytics and data science: winning the game with methods and models. FT Press, Upper Saddle River
Mongiovi M, Bogdanov P, Singh AK (2013) Mining evolving network processes. In: IEEE international conference on data mining, pp 537–546
Obradovic Z (2007) Panathinaikos offense. Fiba Assist Mag 26:33–36
Papadimitriou P, Dasdan A, Garcia-Molina H (2010) Web graph similarity for anomaly detection. J Internet Serv Appl 1(1):19–30
Pei SC, Chen F (2003) Semantic scenes detection and classification in sports videos. In: IPPR conference on computer vision, graphics and image processing (CVGIP), pp 210–217
Pers J, Bon M, Vuckovic G (2006) Cvbase 06 dataset
Perše M, Kristan M, Kovačič S, Vučkovič G, Perš J (2009) A trajectory-based analysis of coordinated team activity in a basketball game. Comput Vis Image Underst 113(5):612–621
Pingali GS, Jean Y, Carlbom I (1998) Real time tracking for enhanced tennis broadcasts. In: Proceedings IEEE computer society conference on computer vision and pattern recognition, pp 260–265

Rayana S, Akoglu L (2016) Less is more: building selective anomaly ensembles. ACM Trans Knowl Discov Data 10(4):42

Rodenberg RM, Feustel ED (2014) Forensic sports analytics: detecting and predicting match-fixing in tennis. J Predict Mark 8(1):77–95

Rozenshtein P, Tatti N, Gionis A (2014) Discovering dynamic communities in interaction networks. In: European conference on machine learning and knowledge discovery in databases, pp 678–693

Sakoe H, Chiba S (1971) A dynamic programming approach to continuous speech recognition. Int Congr Acoust 3:65–69

Shah N, Koutra D, Zou T, Gallagher B, Faloutsos C (2015) Timecrunch: Interpretable dynamic graph summarization. In: ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 1055–1064

Shatkay H, Zdonik SB (1996) Approximate queries and representations for large data sequences. In: IEEE international conference on data engineering, pp 536–545

Sricharan K, Das K (2014) Localizing anomalous changes in time-evolving graphs. In: ACM SIGMOD international conference on management of data, pp 1347–1358

Stensland HK, Gaddam VR, Tennøe M, Helgedagsrud E, Næss M, Alstad HK, Mortensen A, Langseth R, Ljødal S, Landsverk Ø et al (2014) Bagadus: An integrated real-time system for soccer analytics. ACM Trans Multimedia Comput Commun Appl 10(1s):14

Sun J, Faloutsos C, Papadimitriou S, Yu PS (2007) Graphscope: parameter-free mining of large time-evolving graphs. In: ACM SIGKDD international conference on knowledge discovery and data mining, pp 687–696

Thorup M (2000) Near-optimal fully-dynamic graph connectivity. In: ACM symposium on theory of computing, pp 343–350

Travassos B, Davids K, Araújo D, Esteves PT (2013) Performance analysis in team sports: advances from an ecological dynamics approach. Int J Perform Anal Sport 13(1):83–95

Wei X, Sha L, Lucey P, Morgan S, Sridharan S (2013) Large-scale analysis of formations in soccer. In: International conference on digital image computing: techniques and applications, pp 1–8

Zhong D, Chang SF (2001) Structure analysis of sports video using domain models. In: IEEE international conference on multimedia and expo, pp 713–716