Knowledge and Information Systems
https://doi.org/10.1007/s10115-019-01403-9

REGULAR PAPER

Finding events in temporal networks: segmentation meets densest subgraph discovery

Polina Rozenshtein[1,7] · Francesco Bonchi[2,3] · Aristides Gionis[1] · Mauro Sozio[4] · Nikolaj Tatti[5,6]

Received: 5 January 2019 / Revised: 4 September 2019 / Accepted: 13 September 2019

Abstract
In this paper, we study the problem of discovering a timeline of events in a temporal network. We model events as dense subgraphs that occur within intervals of network activity. We formulate the event discovery task as an optimization problem, where we search for a partition of the network timeline into k non-overlapping intervals, such that the intervals span subgraphs with maximum total density. The output is a sequence of dense subgraphs along with corresponding time intervals, capturing the most interesting events during the network lifetime. A naïve solution to our optimization problem has polynomial but prohibitively high running time. We adapt existing recent work on dynamic densest subgraph discovery and approximate dynamic programming to design a fast approximation algorithm. Next, to ensure richer structure, we adjust the problem formulation to encourage coverage of a larger set of nodes. This problem is **NP**-hard; however, we show that on static graphs a simple greedy algorithm leads to approximate solution due to submodularity. We extend this greedy approach for temporal networks, but we lose the approximation guarantee in the process. Finally, we demonstrate empirically that our algorithms recover solutions with good quality.

Keywords Densest subgraph · Segmentation · Dynamic programming · Approximate algorithm

✉ Polina Rozenshtein
polina.rozenshtein@aalto.fi

1 Department of Computer Science, Aalto University, Espoo, Finland

2 ISI Foundation, Turin, Italy

3 Eurecat, Barcelona, Spain

4 Telecom ParisTech University, Paris, France

5 F-Secure Corporation, Helsinki, Finland

6 University of Helsinki, Helsinki, Finland

7 Nordea Data Science Lab, Helsinki, Finland

Published online: 03 October 2019

1 Introduction

Real-world networks are highly dynamic in nature, with new relations (edges) being continuously established among entities (nodes) and old relations being broken. Analyzing the temporal dimension of networks can provide valuable insights about their structure and function; for instance, it can reveal temporal patterns, concept drift, periodicity, temporal events, etc. In this paper, we focus on the problem of *finding dense subgraphs*, a fundamental graph-mining primitive. Applications include community detection in social networks [16,18,48], gene expression and drug interaction analysis in bioinformatics [22,45], graph compression and summarization [21,30,32], spam and security threat detection [13,26], and more.

When working with temporal networks, one has first to define how to deal with the temporal dimension, i.e., how to identify which are the temporal intervals in which the dense structures should be sought. Instead of defining those intervals a priori, in this paper *we study the problem of automatically identifying the intervals that provide the most interesting structures*. We consider a subgraph interesting if it boasts high density. As a result, we are able to discover a sequence of dense subgraphs in the temporal network, capturing the evolution of interesting events that occur during the network lifetime. As a concrete example, consider the problem of *story identification* in online social media [3,8]: The main goal is to automatically discover emerging stories by finding dense subgraphs induced by some entities, such as twitter hashtags, co-occurring in a social media stream.

In our case, we are also interested in finding different stories over the network lifetime. For instance, as one story wanes and another one emerges, one dense subgraph among entities dissipates and another one appears. Thus, by segmenting the timeline of the temporal network into intervals, and identifying dense subgraphs in each interval, we can capture the evolution and progression of the main stories over time. As another example, consider a collaboration network, where a sequence of dense subgraphs in the network can reveal information about the main trends and topics over time, along with the corresponding time intervals.

Challenges and contributions The problem of finding the k densest subgraphs in a static graph has been considered in the literature from different perspectives. One natural idea is to iteratively (and greedily) find and remove the densest subgraphs [49], which unfortunately does not provide any theoretical guarantee. More recent works study the problem of finding k densest graphs with limited overlap, while they provide theoretical guarantees in some cases of interest [7,24]. However, these approaches do not generalize to temporal networks.

For temporal networks, to our knowledge, there are only few papers that consider the task of finding temporally coherent densest subgraphs. The most similar to our work aims at finding a heavy subgraph present in all, or k, snapshots [46]. Another related work focuses on finding a dense subgraph covered by k scattered intervals in a temporal network [44]. Both methods, however, focus on finding a single densest subgraph.

In this paper, instead, we aim at producing a partition of the temporal network that (i) it captures dense structures in the network; (ii) it exhibits temporal cohesion; and (iii) it is amenable to direct inspection and temporal interpretation. To accomplish our objective, we formulate the problem of k- DENSEST- EPISODES (Sect. 2), which requires to find a partition of the temporal domain into k non-overlapping intervals, such that the intervals span subgraphs with maximum total density. The output is a sequence of dense subgraphs along with corresponding time intervals, capturing the most interesting events during the network lifetime.

For example, consider a simple temporal network shown in Fig. 1. It consists of five nodes $\{A, B, C, D, E\}$, which interact at six different time stamps $(1, 2, 4, 5, 7, 8, 10)$. Our goal

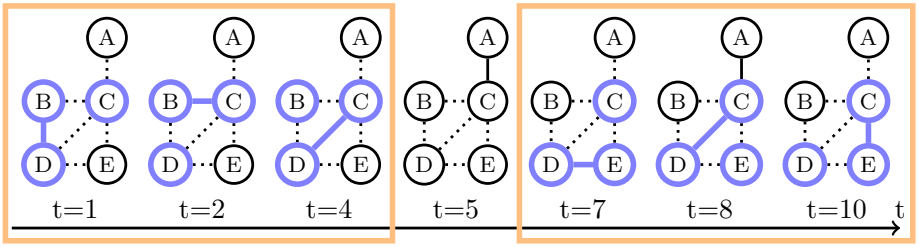

Fig. 1 An example temporal network with five nodes and seven time stamps. The solid lines depict interactions that occur at a given time stamp, while the dotted lines depict interactions that occur at different time stamps. The highlighted time intervals, nodes, and interactions depict events discovered in the network

is to discover time intervals that provide the densest subgraphs. One interesting interval is $I = [7, 10]$, in that four different interactions $\{(A, C), (C, D), (C, E), (D, E)\}$ occur during I, with three of them constructing a prominently dense subgraph—a clique $\{C, D, E\}$. Thus, a pair (interval, a subgraph covered by the interval) ($[7, 10]$, $\{(C, D), (C, E), (D, E)\}$) summarizes an interesting episode in the history of interactions of this toy network. Another interesting interval is $[1, 4]$ as it contains a clique $\{B, C, D\}$. Thus, our network partition would be $(([1, 4], \{(B, C), (B, D), (D, E)\}), ([7, 10], \{(C, D), (C, E), (D, E)\}))$. Note that interaction (A, C) is completely ignored as it does not contribute to any dense subgraph.

A naïve solution to this problem has polynomial but prohibitively high running time. Thus, we adapt existing recent work on dynamic densest subgraph discovery [19] and approximate dynamic programming [47] to design a fast approximation algorithm (Sect. 3).

Next (Sect. 4), we shift our attention to encouraging coverage of a larger set of nodes, so as to produce richer and more interesting structures. The resulting new problem formulation turns out to be **NP**-hard. However, on static graphs a simple greedy algorithm leads to approximate solution due to the submodularity of the objective function. Following this observation, we extend this greedy approach for the case of temporal networks. Despite the fact that the approximation guarantee does not carry on when generalizing to the temporal case, our experimental evaluation indicates that the method produces solutions of very high quality.

Experiments on synthetic and real-world datasets (Sect. 5) and a case study on Twitter data (Sect. 6) confirm that our methods are efficient and produce meaningful and high-quality results.

2 Problem formulation

We are given a *temporal graph* $G = (V, \mathcal{T}, E)$, where V denotes the set of nodes, $\mathcal{T} = [0, 1, \ldots, t_{\max}] \subset \mathbb{N}$ is a discrete time domain, and $E \subseteq V \times V \times \mathcal{T}$ is the set of all temporal edges. Given a temporal interval $T = [t_1, t_2]$ with $t_1, t_2 \in \mathcal{T}$, let $G[T] = (V[T], E[T])$ be the subgraph induced by the set of temporal edges $E[T] = \{(u, v) \mid (u, v, t) \in E, t \in T\}$ with $V[T]$ being the set of endpoints of edges $E[T]$.

Definition 1 (*Episode*) Given a temporal graph $G = (V, \mathcal{T}, E)$, we define an episode as a pair (I, H) where $I = [t_1, t_2]$ is a temporal interval with $t_1, t_2 \in \mathcal{T}$ and H is a subgraph of $G[I]$.

Our goal is to find a set of interesting episodes along the lifetime of the temporal graph. In particular, our measure of interestingness is the density of the subgraph in the episodes. We adopt the widely used notion of density of a subgraph $H = (V(H), E(H))$ as the average degree of the nodes in the subgraph, i.e., $d(H) = \frac{|E(H)|}{|V(H)|}$. While several definitions for density have been studied in the literature, the one we focus on enjoys the following nice properties: It can be optimized exactly [27] and approximated efficiently [15], while a densest subgraph can be computed in real-world graphs containing up to tens of billions of edges [17].

Problem 1 (k-Densest-Episodes) Given a temporal graph $G = (V, \mathcal{T}, E)$ and an integer $k \in \mathbb{N}$, find a set of k episodes $S = \{(I_\ell, H_\ell)\}$, for $\ell = 1, \ldots, k$ such that $\{I_\ell\}$ are disjoint intervals and the profit $\sum_{\ell=1}^{k} d(H_\ell)$ is maximized.

We can solve Problem 1 in polynomial time. To see this, let S^* be an optimum solution for Problem 1 and let $\mathcal{I}(S^*) = \{I_1, \ldots, I_k\}$ and $\mathcal{G}(S^*) = \{H_1, \ldots, H_k\}$. Observe that without loss of generality, we can assume that the union of the intervals in $\mathcal{I}(S^*)$ is equal to the set of time stamps $\mathcal{T}$, that is, $\mathcal{I}(S^*)$ is a k-segmentation of $\mathcal{T}$. This follows from the fact that by increasing the length of the I_ℓ's, the density of the corresponding densest subgraphs cannot decrease.

Given an interval I_ℓ, a densest subgraph in $G(I_\ell)$ can be found by running any algorithm for computing a densest subgraph: in $\mathcal{O}(nm \log n)$ time by the easy-to-implement algorithm of Goldberg et al. [27,43] or in $\mathcal{O}(nm \log(n^2/m))$ time by the more involved algorithm by Gallo et al. [25], where n and m denote the number of nodes and edges in $G(I_\ell)$, respectively. An optimal segmentation can be solved by a standard dynamic-programming approach, requiring $\mathcal{O}(k|\mathcal{T}|^2)$ steps [9]. By combining the subroutine for computing an optimal segmentation with either subroutine for computing a densest subgraph for each given interval, one can find a solution to Problem 1 in $\mathcal{O}(k|\mathcal{T}|^2 nm \log n)$, or $\mathcal{O}(k|\mathcal{T}|^2 nm \log(n^2/m))$, respectively.

As a post-processing step, we can trim the intervals in an optimal solution $S^* = \{(I_\ell, H_\ell)\}$ by calculating the minimum subinterval of I_ℓ, which spans all edges of H_ℓ, for each $\ell = 1, \ldots, k$.

3 Approximate dynamic programming

The simple algorithm discussed in the previous section has a running time, which is prohibitively expensive for large graphs. In this section, we develop a fast algorithm with approximation guarantees.

The derivations below closely follow the ones in [47], which improves [29]. However, we cannot use those results directly: Both papers work with minimization problems and use the fact that the profit of an interval is not less than the profit of its subintervals. In contrast, our problem is a maximization problem and requires a tailored solution.

Given a time interval $T = [t_1, t_2]$, we write $d^*(T)$ to denote the density of the densest subgraph in T, that is, $d^*(T) = \max_{H \subseteq G(T)} d(H)$. For simplicity, we define $d^*([t_1, t_2]) = 0$ if $t_2 < t_1$. Problem 1 is now a classic k-segmentation problem of $\mathcal{T}$ maximizing the total sum of scores $d^*(T)$ for individual time intervals. For notation simplicity, we assume that all time stamps $\mathcal{T}$ are enumerated by integers from 1 to r.

Let $o[i, \ell]$ be the profit of the optimal ℓ-segmentation using only the first i time stamps. Then,

$$o[i, \ell] = \max_{j<i} o[j, \ell - 1] + d^*(j + 1, i), \tag{1}$$

Algorithm 1: `ApproxDP`(k, ϵ) computes k-segmentation with ϵ-approximation guarantee

Input: number of intervals k, parameter ϵ
Output: approximate solution $s[i, \ell]$ for $i \in [1, r]$, $\ell \in [1, k]$

```
1 for i = 1, ..., r do s[i, 1] = d*([1, i]) for ℓ = 2, ..., k do
2 |  A = [ ];
3 |  for i = 1, ..., r do
4 |  |  add i to A;
5 |  |  s[i, ℓ] = max{s[i − 1, ℓ], s[i, ℓ − 1], max_{a∈A}(s[a − 1, ℓ − 1] + d*([a, i]))};
6 |  |  A = SPRS(A, s[i, ℓ], ℓ, ϵ)
7 |  end
8 end
9 return s
```

Algorithm 2: `SPRS`$(A, \sigma, \ell, \epsilon)$, a subroutine keeping the candidate list short.

Input: current enumerated candidates $A = (a_1, a_2, \ldots, a_{|A|})$, sparsification factor $\sigma = s[i, \ell]$, current number of intervals ℓ, approximation parameter ϵ
Output: sparsified A

```
1 δ = σ ϵ/(k+ℓϵ);
2 j = 1;
3 while j < |A| − 1 do
4 |  if s[a_{j+2}, ℓ − 1] − s[a_j, ℓ − 1] ≤ δ then remove a_{j+1} from A else j = j + 1
5 end
6 return A
```

and $o[i, k]$ can be computed recursively.

Our goal is to approximate $o[i, \ell]$ quickly with a score which we will denote by $s[i, \ell]$. The main idea behind the speedup is not to test all possible values of j in Eq. 1. Instead, we are going to keep a small set of candidates, denoted by A, and only use those values for testing.

The challenge is how to keep A small enough while at the same time guarantee the approximation ratio. The pseudo-code achieving this balance is given in Algorithm 1, while a subroutine that keeps the candidate list short is given in Algorithm 2. Algorithm 1 executes a standard dynamic programming search: It assumes that partition of $i' < i$ first data points into $\ell - 1$ intervals is already calculated and finds the best last interval $[a, i]$ for partitioning of i first points into l intervals. However, it does not consider all possible candidates $[a, i]$, but only a sparsified list, which guarantees to preserve a quality guarantee. The sparsified list is built for a fixed number of intervals ℓ starting from empty list. Intuitively, it keeps only candidates $A = \{a_j\}$ with significant difference in $s[a_j, \ell - 1]$. The significance of the difference depends on the current best profit $s[i, \ell]$: The larger the value of the solution found, the less cautious we can be about lost candidates and the coarser becomes A. Thus, we need to refine A by Algorithm 2 after each processed i.

We first study the approximation guarantee of `ApproxDP`, assuming that $d^*(\cdot)$ is calculated exactly.

Proposition 1 *Let $s[i, \ell]$ be the profit table constructed by `ApproxDP`(k, ϵ). Then, $s[i, \ell](\frac{\ell\epsilon}{k} + 1) \geq o(i, \ell)$.*

To prove the proposition, let us first fix ℓ and let A_i be the set of candidates in A to be tested on line 6 of round i. Let δ_i be the value of δ in Algorithm 2, called on iteration i. Then, δ_{i-1} is the coarsening parameter used to sparsify A_i.

Lemma 1 *For every $b \in [1, i]$, there is $a_j \in A_i$, such that*

$$s[a_j - 1, \ell - 1] + d^*([a_j, i]) \geq s[b - 1, \ell - 1] + d^*([b, i]) - \delta_{i-1}.$$

Proof We say that a list of numbers $A = (a_j)$ is i-dense, if

$$s[a_{j+1} - 1, \ell - 1] - s[a_j - 1, \ell - 1] \leq \delta_{i-1} \text{ or } a_{j+1} = a_j + 1,$$

for every $a_j \in A$ with $j < |A|$. We first prove by induction over i that A_i is i-dense.

Assume that A_{i-1} is $(i-1)$-dense. The `SPRS` procedure never deletes the last element, so $(i-1) \in A_{i-1}$, and $A_{i-1} \cup \{i\}$ is $(i-1)$-dense. Note that $\delta_{i-2} \leq \delta_{i-1}$, because $s[i, \ell]$ is monotone, and $s[i, \ell] \geq s[i-1, \ell]$, due to explicit check in line 6 of procedure `ApproxDP`. Thus, $A_{i-1} \cup \{i\}$ is i-dense. Since $A_i = \texttt{SPRS}(A_{i-1}) \cup \{i\}$, and `SPRS` does not create gaps larger than δ_{i-1}, the list A_i is i-dense.

Let a_j be the largest element in A_i, such that $a_j \leq b$. Then, either $a_j \leq b < a_{j+1}$ or $b = a_{|A_i|}$ and $a_j = a_{|A_i|}$. In the first case, due to monotonicity, we have $s[a_{j+1}, \ell - 1] \geq s[b, \ell - 1]$, which gives $s[b-1, \ell-1] - s[a_j - 1, \ell - 1] \leq \delta_{i-1}$. The second case is trivial.

Due to monotonicity, $d^*([a_j, i]) \geq d^*([b, i])$. This concludes the proof. □

We can now complete the proof of Proposition 1.

Proof of Proposition 1 We will prove the result with induction over ℓ. The claim holds for $\ell = 1$ and any i as we initialize $s[i, 1]$ by optimal values (on line 1 of Algorithm 1). We assume that the approximation guarantee holds for $\ell - 1$, that is,

$$s[i, \ell - 1](1 + \frac{\epsilon}{k}(\ell - 1)) \geq o[i, \ell - 1]$$

and we prove the result for ℓ.

Let $\alpha = (1 + \frac{\epsilon}{k}(\ell - 1))$. Let b be the starting point of the last interval of optimal solution $o[i, \ell]$, and let a_j be as given by Lemma 1. We upper bound

$$\delta_{i-1} = s[i-1, \ell-1]\frac{\epsilon}{k + \epsilon\ell} \leq s[i, \ell]\frac{\epsilon}{k + \epsilon\ell} \leq s[i, \ell]\frac{\epsilon}{\alpha k}. \tag{2}$$

Then,

$$\begin{aligned}
\alpha s[i, \ell] &\geq \alpha(s[a_j - 1, \ell - 1] + d^*([a_j, i])) && (a_j \in A_i)\\
&\geq \alpha(s[b - 1, \ell - 1] + d^*([b, i]) - \delta_{i-1}) && \text{(Lemma 1)}\\
&= \alpha s[b - 1, \ell - 1] + \alpha d^*([b, i]) - \alpha\delta_{i-1}\\
&\geq o[b - 1, \ell - 1] + \alpha d^*([b, i]) - \alpha\delta_{i-1} && \text{(induction)}\\
&\geq o[b - 1, \ell - 1] + d^*([b, i]) - \alpha\delta_{i-1} && (\alpha \geq 1)\\
&\geq o[b - 1, \ell - 1] + d^*([b, i]) - s[i, \ell]\frac{\epsilon}{k} && (Eq.\ 2)\\
&= o[i, \ell] - s[i, \ell]\frac{\epsilon}{k}.
\end{aligned}$$

As a result, $s[i, \ell](1 + \frac{\epsilon}{k}\ell) \geq o[i, \ell])$. □

Let us now address the running time of the approximate dynamic programming.

Proposition 2 *The running time of `ApproxDP` is $\mathcal{O}(\frac{k^2}{\epsilon}r)$.*

Proof Let us fix i and ℓ, and count the number of candidates in A_i. Note that $|A_i| = |\texttt{SPRS}(A_{i-1})| + 1$. The list of candidates $\texttt{SPRS}(A_{i-1})$ corresponds to a monotonically increasing sequence of $s[a, \ell]$, with consecutive elements being at least δ_{i-1} apart. Thus, $|\texttt{SPRS}(A_{i-1})| \leq \frac{s[i-1,\ell]}{\delta_{i-1}} = \frac{k+\ell\epsilon}{\epsilon} \leq \frac{k(1+\epsilon)}{\epsilon}$ and the number of operations in one call of the inner loop (lines 4–8) of Algorithm 1 is $\mathcal{O}(k/\epsilon)$. Since this loop is called kr times, the result follows. □

Since computing d^* requires time $\mathcal{O}(nm \log n)$, the total running time is $\mathcal{O}(r\frac{k^2}{\epsilon} nm \log n)$, where $r = |\mathcal{T}|$. We further speed up our algorithm by approximating the value d^* by means of one of the approaches developed by [19]. In particular, we employ the algorithm that maintains a $2(1+\epsilon)$-approximate solution for the incremental densest subgraph problem (i.e., edge insertions only), while having poly-logarithmic amortized cost. We shall refer to such an algorithm as `ApprDens`.

`ApprDens` allows us to efficiently maintain the approximate density of the densest subgraph $d^*([a, i])$ for each a in A_i in `ApproxDP`, as larger values of i are processed and edges are added. Whenever we remove an item a from A_i in `SPRS`, we also drop the corresponding instance of `ApprDens`.

From the fact that an approximate densest subgraph can be maintained with poly-logarithmic amortized cost, it follows that our algorithm has quasi-linear running time.

Proposition 3 *`ApproxDP` combined with `ApprDens` runs in $\mathcal{O}(\frac{k^2}{\epsilon_1\epsilon_2^2}|\mathcal{T}|m_t \log^2 n)$ time, where ϵ_1 and ϵ_2 are the respective approximation parameters for `ApproxDP` and `ApprDens` and m_t is the maximum number of edges per time stamp.*

For real-world highly dynamic temporal networks, we can safely assume that m_t is a small constant.

Proof To fill in cell $s[i, l]$, we need to update $|A_i| = \mathcal{O}(k/\epsilon)$ graphs by adding at most (some edges can be already in the graphs) m_i edges—the number of edges with t_i time stamp. Let m_t be the maximum number of edges per time stamp. Theorem 4 in [19] states that maintaining the graph with m_i edges requires $\mathcal{O}(m_i\epsilon_2^{-2} \log^2 n)$ time. We still need to fill $k|\mathcal{T}|$ cells in the DP matrix. Combining these two results proves the proposition. □

When combining `ApproxDP` with `ApprDens`, we wish to maintain the same approximation guarantee of `ApprDens`. Recall that `ApproxDP` leverages the fact that the profit function is monotone and non-increasing. Unfortunately, `ApprDens` does not necessarily yield a monotone score function, as the density of the computed subgraph might decrease when a new edge is inserted. This can be easily circumvented by keeping track of the best solution, i.e., the subgraph with highest density. The following proposition holds.

Proposition 4 *`ApproxDP` combined with `ApprDens` yields a $2(1+\epsilon_1)(1+\epsilon_2)$-approximation guarantee.*

Proof Let $d_a^*(T)$ be the density of the graph returned by `ApprDens` for a time interval T. Let O be the optimal k-segmentation, let $\mathcal{I}(O)$ be the intervals of this solution, and let $q_1 = \sum_{I\in\mathcal{I}(O)} d^*(I)$ be its score. Let also $q_2 = \sum_{I\in\mathcal{I}(O)} d_a^*(I)$. Let q_3 be the score of the optimal k-segmentation O_a using d_a^*. Note that the intervals constituting the solution O_a may not be the same as in O, as they are optimal solutions for different interval scoring functions d^* and d_a^*. Thus, q_3 may not be equal to q_2. Let q_4 be the score of the segmentation produced by `ApproxDP`. Then,

$$q_1 \leq 2(1+\epsilon_2)q_2 \leq 2(1+\epsilon_2)q_3 \leq 2(1+\epsilon_2)(1+\epsilon_1)q_4,$$

completing the proof. □

We will refer to this combination of ApproxDP with ApprDens as Algorithm kGAPPROX.

4 Encouraging larger and more diverse subgraphs

Problem 1 is focused on total density maximization; thus, its solution can contain graphs which are dense, but union of their node sets can cover only a small part of the network. Such segmentation is useful when we are interested in the densest temporally coherent subgraphs, which can be understood as tight cores of temporal clusters. However, segmentations with larger but less dense subgraphs, covering a larger fraction of nodes, can be useful to get a high-level explanation of the whole temporal network. To allow for such segmentations, we extend Problem 1 to take into account node coverage.

Denote the set of subgraphs G_i, which are included in solution episodes $S = \{(I_i, G_i)\}$ as $\mathcal{G} = \{G_i\}$ for $i = 1, \ldots, k$. Given a collection of subgraphs $\mathcal{G}$, let $x_v(\mathcal{G}) = |\{G_i \in \mathcal{G} : v \in V(G_i), G_i \in \mathcal{G}\}|$ be the number of subgraphs in $\mathcal{G}$, which include node v. We consider generalized cover functions of the type

$$\text{cover}(\mathcal{G} \mid w) = \sum_{v \in V} w(x_v(\mathcal{G})),$$

where w is a nonnegative non-decreasing concave function of $x_v(\mathcal{G})$. If $w(x_v(\mathcal{G}))$ is a 0–1 indicator function, then the function cover$(\mathcal{G} \mid w)$ is a standard cover, which is intuitive and easy to optimize by a greedy algorithm. Another instance of the generalized cover function, inspired by text summarization research [35], is $w(x_v(\mathcal{G})) = \sqrt{x_v(\mathcal{G})}$. It ensures that the marginal gain of a node decreases proportionally to the number of times the node is covered. We add the cover term to the cost function of Problem 1, and we obtain the resulting problem formulation.

Problem 2 (k-Densest-Episodes-EC) Given a temporal graph $G = (V, \mathcal{T}, E)$, integer k, parameter $\lambda \geq 0$, find a k-segmentation $S = \{(I_i, G_i)\}$ of G, such that profit$(S) = \sum_{G_i \in \mathcal{G}} d(G_i) + \lambda \, \text{cover}(\mathcal{G} \mid w)$ is maximized.

Unlike Problem 1, this problem cannot be solved in polynomial time.

Proposition 5 *Problem 2 is **NP**-hard.*

Proof We will prove the hardness by reducing the set packing problem to k- DENSEST- EPISODES- EC. In the set packing problem, we are given a collection $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ of sets and are asked whether there are p disjoint sets. We can safely assume that $|C_i| = 3$.

Assume that we are given such a collection, and let us construct the temporal graph. The nodes V consist of two sets V_1 and V_2. The first set V_1 corresponds to the elements in $\bigcup_i C_i$. The second set V_2 consists of $q = 6\ell + 3$ nodes. There are 2ℓ time stamps. At the $2i$th time stamp, we connect the nodes corresponding to C_i, while at odd time stamps, we full-connect V_2. Finally, we set $k = \ell + p$ and $\lambda = 1/(|V| + 1)$. We use 0–1 indicator function for w.

We claim that there is a solution to the set packing problem if and only if there is a solution to k- DENSEST- EPISODES- EC with the profit of at least $\ell(q-1)/2 + p + \lambda(3p + q)$.

To prove the only if direction, assume there is a collection $\mathcal{C}'$ of p disjoint sets. Build a k-segmentation by selecting each clique spanning V_2 to be in its own segment, as well as the three cliques corresponding to the sets in $\mathcal{C}'$. This solution will have the necessary profit.

Let us now prove the if direction. Assume an optimal k-segmentation S. It is easy to see that if the ith segment contains an odd time stamp, then G_i must be the clique spanning V_2. On the other hand, if the ith segment is equal to $[2j, 2j]$, then G_j is a clique connecting C_j.

Let a be the number of segments containing odd time stamps, we can safely assume that $a > 0$. Let b be the number of segments containing only even time stamps. Let c be the total number of nodes in V_1 covered by at least one segment. Then,

$$\text{profit}(S) = a(q-1)/2 + b + \lambda(c+q).$$

We assume that $\text{profit}(S) \geq \ell(q-1)/2 + p + \lambda(3p+q)$. Since $b + \lambda(c+q) \leq \ell + 1 < (q-1)/2$ and $\lambda(c+q) < 1$, this is only possible if $a = \ell$, $b = p$, and $c = 3p$. This completes the proof. □

4.1 *k* static overlapping densest subgraphs

Given the complexity of Problem 2, we start with analysis of a static graph case. We formulate the *k-overlapping-densest-subgraphs problem* and design a linear algorithm with an approximation guarantee. We will later apply the developed approach to temporal graphs; however, the algorithm can be used as an efficient stand-alone method for finding overlapping dense subgraphs.

Problem 3 (*k static overlapping densest subgraphs*) Given a static graph $H = (V, E')$, integer k, and real $\lambda \geq 0$, find a set of k subgraphs $\mathcal{H} = \{H_i \subseteq H\}$, such that $\text{profit}_{ST}(\mathcal{H}) = \sum_{H_i \subseteq \mathcal{H}} d(H_i) + \lambda \cdot \text{cover}(\mathcal{H} \mid w)$ is maximized.

Next, we show below how to obtain a constant-factor approximate solution. We start with showing that the generalized cover function has beneficial combinatorial properties: It is submodular, nonnegative, and non-decreasing with respect to the set of subgraphs. The density term of the cost function of Problem 2 (and Problem 3) is a linear function of subgraphs, and thus the whole cost function is nonnegative, non-decreasing, and submodular.

Proposition 6 *Function* $\text{cover}(\mathcal{G} \mid w)$ *is a nonnegative, non-decreasing, and submodular function of subgraphs.*

Proof For a fixed $v \in V$ function $x_v(\mathcal{G})$ is non-decreasing modular (and submodular): for any set of subgraphs X and a new subgraph x holds that $x_v(X \cup \{x\}) - x_v(X) = 1$ if v belongs to x and does not belong to any subgraph in X, otherwise 0. By the property of submodular functions, composition of concave non-decreasing and submodular non-decreasing is non-decreasing submodular. Function $\text{cover}(\mathcal{G} \mid w)$ is submodular non-decreasing as a nonnegative linear combination. Nonnegativity follows from nonnegativity of w. □

To solve Problem 3, we can search greedily over subgraphs. Let $\mathcal{H}_{i-1} = \{H_1, \ldots, H_{i-1}\}$, and define marginal node gain, given weight function w, as

$$\delta(v \mid \mathcal{H}_{i-1}, w) = w(x_v(\mathcal{H}_{i-1} \cup \{v\})) - w(x_v(\mathcal{H}_{i-1})). \tag{3}$$

Here, $\{v\}$ refers to a graph containing only v. Then, denote the marginal gain of subgraph H_i given already selected graphs $\mathcal{H}_{i-1}$ as

$$\chi(H_i \mid \mathcal{H}_{i-1}, w) = d(H_i) + \lambda \sum_{v \in H_i} \delta(v \mid \mathcal{H}_{i-1}, w). \tag{4}$$

Greedy algorithm for Problem 3 consequently builds the set $\mathcal{H}$ by adding H_i, which maximizes gain $\chi(H_i \mid \mathcal{H}_{i-1})$. If we can find H_i optimally, such algorithm yields $1 - 1/e$ approximation due to submodular maximization with cardinality constrains (see [42] for this classic result).

To find the optimal H_i, we need to solve the following problem.

Problem 4 Given a static graph $H = (V, E')$, a set of subgraphs $\mathcal{H}_{i-1} = \{H_1, \ldots, H_{i-1}\}$, find a graph $F \subseteq H$, such that $\chi(F \mid \mathcal{H}_{i-1})$ is maximized.

Luckily, Problem 4 can be transformed into a (weighted) densest subgraph problem. In order to do so, we will define a weighted fully connected graph $R = (V, V \times V, a)$ having the same nodes V as H with the weights $a(u, v)$ defined as

$$a(u, v) = I[(u, v) \in E'] + \frac{\lambda}{1 + I[u = v]}(\delta(u \mid \mathcal{H}_{i-1}, w) + \delta(v \mid \mathcal{H}_{i-1}, w)).$$

Here, $I[\cdot]$ is an indicator function, returning 1 if the condition is true, and 0 otherwise. Note that we allow self-loop edges. Let R' be a subgraph in R and let F be the induced subgraph in H having the same nodes as R'. Then, it is now straightforward to see that

$$\chi(F \mid \mathcal{H}_{i-1}, w) = d(R').$$

In other words, solving Problem 4 is equivalent to solving densest subgraph problem in R. Consequently, we can solve Problem 4 exactly in $\mathcal{O}(|V|^3)$ time [25]. Alternatively, we can estimate it efficiently with 1/2-approximation in $O(|V|^2)$ time by Charikar et al. [15]. We will use the latter algorithm and refer to it as `StaticGreedy`.

Now we have everything to design and analyze an approximation algorithm for Problem 3. Algorithm 3 greedily finds k subgraphs to solve Problem 3.

Each subgraph is sought with 1/2-approximation guarantee, and due to submodularity, greedy optimal subgraph search would be a $(1-1/e)$-approximation. Combining these results leads to the following statement.

Proposition 7 *Algorithm 3 is a* $1/2(1 - 1/e) \approx 0.31606$ *approximation for Problem 3.*

Proof Let y be the value of $profit_{ST}$ score of k greedily sought subgraph, assuming that each subgraph was sought optimally. The ith subgraph has a marginal gain y_i, thus $y = \sum_i^k y_i$. Let optimal solution of Problem 3 be y^*. Due to greedy submodular optimization $y \geq (1-1/e)y^*$, Algorithm 3 uses 1/2-approximation algorithm `StaticGreedy` for subgraph search, thus $\bar{y}_i \geq y_i/2$, where $\bar{y}_i$ is the marginal gain of the i-th subgraph included into the solution. Let $\bar{y}$ be the value of final solution output by Algorithm 3. Putting everything together, we have $\bar{y} = \sum_i^k \bar{y}_i \geq \sum_i^k y_i/2 = y/2 \geq 1/2(1 - 1/e)y^*$. This concludes the proof. □

The running time of Algorithm 3 is defined by the running time of the greedy subroutine and is $\Theta(k|V|^2)$.

4.2 Greedy dynamic programming

Similarly to Problem 1, we will use dynamic programming for Problem 2. However, as the problem is hard, we have to rely on greedy choices of the subgraphs. Thus, the obtained solution does not have any quality guarantee.

Let $M[\ell, i]$ be the profit of i first points into ℓ intervals, let $C[\ell, i]$ be the set of subgraphs $\mathcal{G}_\ell = \{G_1, \ldots, G_\ell\}$ selected on these ℓ intervals, $1 \leq \ell \leq k$ and $0 \leq i \leq m$.

Algorithm 3: `StaticKDensest`

Input: static graph $H = (V, E')$, integer k, parameter $\lambda \geq 0$
Output: a set of k subgraphs $\mathcal{H} = \{H_j \subseteq H\}$

$\mathcal{H} = \emptyset$;
for $j = 1, \ldots, k$ **do**
  $H_j = F$ /* where F is a solution of Problem 4 for $H = (V, E')$ and $\mathcal{H}_{i-1} = \mathcal{H}$ */
  $\mathcal{H} = \mathcal{H} \cup \{H_j\}$;
end
return $\mathcal{H}$

Define marginal gain interval $[j, i]$, given that $j - 1$ are already segmented into $\ell - 1$ interval, (here χ is defined in Eq. 4):

$$\text{gain}([j, i], C[\ell - 1, j - 1]) = \max_{G' \subseteq G([j,i])} \chi(G' \mid C[\ell - 1, j - 1]). \tag{5}$$

This leads to a dynamic program

$$M[\ell, i] = \max_{1 \leq j \leq i+1} M[\ell - 1, j - 1] + \text{gain}([j, i], C[\ell - 1, j - 1]) \text{ for } 1 < \ell \leq k,$$
$$M[1, i] = d^*([0, i]) \text{ for } 0 \leq i \leq m,$$
$$M[k', 0] = 0 \text{ for } 1 \leq k' \leq k.$$

After filling this table, $M[k, m]$ contains the profit of k-segmentation with subgraph overlaps. $C[k, m]$ will contain selected subgraphs, and the intervals and subgraphs can be reconstructed, if we keep track of the starting points of selected last intervals. Note that profit $M[k, m]$ is not optimal, because the choice of subgraph G_i depends on the interval and the previous choices.

We perform dynamic programming by approximation algorithm `ApproxDP`, and the densest subgraph for each candidate interval is retrieved by Epasto et al. [19]. We refer to the resulting algorithm as KGCVR.

To keep track on number of x_v when we construct $\mathcal{G}$, we need to keep frequencies of each node. To avoid extensive memory costs, in the experiments we use Min-Count sketches.

5 Experimental evaluation

We evaluate the performance of the proposed algorithms on synthetic graphs and real-world social networks. The datasets are described below. Unless specified, we post-process the output of all algorithms and report the optimal densest subgraphs in the output intervals. Our datasets and implementations are publicly available.[1]

5.1 Synthetic data

We generate a temporal network with k planted communities and a background network. All graphs are Erdős-Rényi. The communities G' have the same density, disjoint set of nodes, and are planted in consecutive non-overlapping intervals. The background network G includes

[1] https://github.com/polinapolina/segmentation-meets-densest-subgraph.

nodes from all planted communities G'. The edges of G are distributed uniformly on the timeline. In a typical setup, the length of the whole time interval T is $|T| = 1000$ time units, while the edges of each G' are generated in intervals of length $|T'| = 100$ time units. The densities of the communities and the background network vary. The number of nodes in G is set to 100.

We produced two families of synthetic temporal networks: *Synthetic1* and *Synthetic2*. In the first setting (dataset family *Synthetic1*), we vary the average degree of the background network from 0.5 to 4 and fix the density of the planted 5-cliques to 4. *Synthetic1* allows to test the robustness of our algorithms against background noise. In the second setting (dataset family *Synthetic2*), we vary the density of planted eight-node graphs from 2 to 7, while the average degree of the background network is fixed to 2. A separate synthetic dataset *Synthetic3* is designed to test the effect of setting different parameters k in the algorithms. The dataset contains $k = 10$ intervals with the activity of eight-node subgraphs with average degree 5, and the background noise has average degree 2.

5.2 Real-world data

We use the following real-world datasets: *Facebook* [51] is a subset of Facebook activity in the New Orleans regional community. Interactions are posts of users on each other walls. The data cover the time period from 9.05.06 to 20.08.06. The *Twitter* dataset tracks activity of Twitter users in Helsinki in year 2013. As interactions, we consider tweets that contain mentions of other users. The *Students*[2] dataset logs activity in a student online network at the University of California, Irvine. Nodes represent students, and edges represent messages with ignored directions. *Enron*:[3] is a popular dataset that contains e-mail communication of senior management in a large company and spans several years.

For a case study, we create a hashtag network from Twitter dataset (the same tweets from users in Helsinki in year 2013): Nodes represent hashtags—there is an interaction, if two hashtags occur in the same tweet. The time stamp of the interaction corresponds to the time stamp of the tweet. We denote this dataset as *Twitter#*.

5.3 Optimal baseline

A natural baseline for kGAPPROX is OPTIMAL, which combines exact dynamic programming with finding the optimal densest subgraph for each candidate interval. Due to the high running time of OPTIMAL, we generate a very small dataset with 60 time stamps, where each time stamp contains a random graph with 3–6 nodes and random density. We vary the number of intervals k and report the value of the solution (without any post-processing) and the running time in Fig. 2. On this toy dataset, kGAPPROX is able to find near-optimal solution, while being significantly faster than OPTIMAL.

5.4 Results on synthetic datasets

Next, we evaluate the performance of kGAPPROX on the synthetic datasets *Synthetic1* and *Synthetic2* by assessing how well the algorithm finds the planted subgraphs. We report mean

[2] http://toreopsahl.com/datasets/#online_social_network.

[3] http://www.cs.cmu.edu/~./enron/.

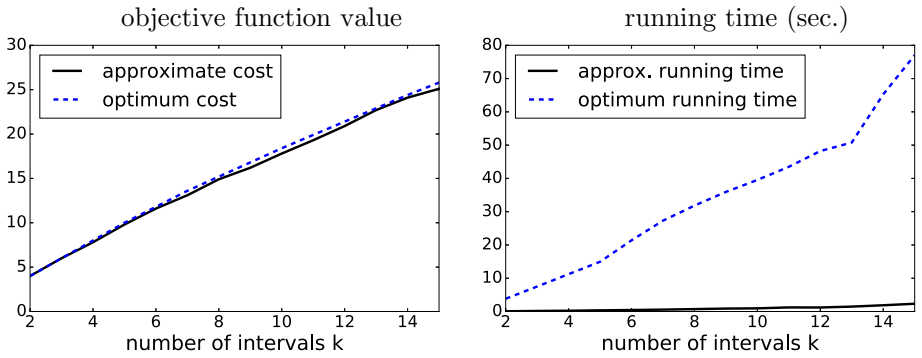

Fig. 2 Comparison between optimum and approximate solutions (OPTIMAL and KGAPPROX). Approximate algorithm was run with $\epsilon_1 = \epsilon_2 = 0.1$. Running time is in seconds

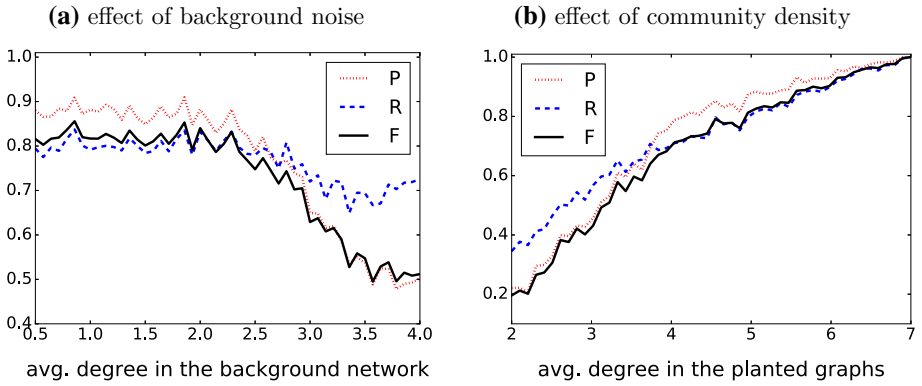

Fig. 3 Precision, recall, and F-measure on synthetic datasets. For plot **a**, the community average degree is fixed to 5 (*Synthetic1* dataset), and for plot, **b** the background network degree is fixed to 2 (*Synthetic2* dataset). Plot **a** the mean standard deviation for precision is 0.193, for recall is 0.183, and for F-measure is 0.180. Plot **b** the mean standard deviation for precision is 0.188, for recall is 0.178, and for F-measure is 0.173

precision, recall, and F-measure, calculated with respect to the ground-truth subgraphs. All results are averaged over 100 independent runs.

First, Fig. 3a depicts the quality of the solution as a function of background noise. Recall that the *Synthetic1* dataset contains planted eight-node subgraphs with average degree 5. Precision and recall are generally high for all values of average degree in the background network. However, precision degrades as the density of the background network increases, as then it becomes cost-beneficial to add more nodes in the discovered densest subgraphs.

Second, Fig. 3b shows the quality of the solution of KGAPPROX as a function of the density in the planted subgraphs. Note that, in *Synthetic2* the density of the background network is 2. Similarly to the previous results, the quality of the solution, especially recall, degrades much only when the density of the planted and the background network becomes similar.

Figure 4 demonstrates how well the true event intervals are recovered in the case of a synthetic *Synthetic3* dataset with $k = 10$ planned events intervals. The true value of k was treated as unknown, and KGAPPROX was run with all possible integer values of k in [2, 20].

Figure 4a shows the quality of the intervals, precision, and recall are calculated with respect to the length of the overlap between the true interval and the output one.

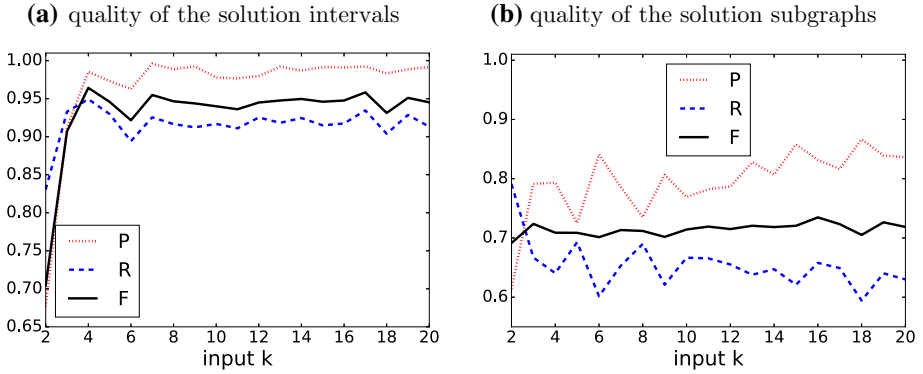

Fig. 4 Quality of the solutions in the case of unknown k. Planted $k = 10$ intervals with eight-node subgraphs each (*Synthetic3* dataset). Plot **a** shows the quality of the solution segmentation, and plot **b** shows the quality of the subgraphs sought in the intervals

Since the number of intervals in the segmentation and the ground truth is different, we compare each output interval to its best match in terms of F-measure. That is, let $(I_1, \ldots, I_k)$ and $(I'_1, \ldots, I'_{k'})$ be the set of ground-truth intervals and solution intervals with k not necessarily be equal to k'. For each I'_i in the solution, we find the best matching interval in the ground truth $I^*_i = \max_{I_i \in (I_1, \ldots, I_k)} F(I'_i, I_i)$. Here, F is F-measure, with precision and recall being calculated with respect to time stamps: the number of time stamps from the ground-truth interval I_i, which also belong to the interval I'_i, divided by the number of time stamps in I'_i (precision) or divided by the number of time stamps in I_i (recall). Once such matching interval I^*_i is found for each I'_i, we calculate and report precision $P(I^*_i, I_i)$, recall $R(I^*_i, I_i)$, and F-measure $F(I^*_i, I_i)$, defined with respect to the time stamps as described above.

All the reported measures are averaged over the output intervals (and over 100 runs). After matching the intervals, we also evaluate the quality of the densest subgraphs and compare their node sets to the ground-truth events in the corresponding intervals (Fig. 4b). As we can see, the intervals are in general recovered quite well, even though the algorithm is given an incorrect value of k. The quality of the subgraph recovery is generally lower, which is the results of shifted borders of the intervals.

5.5 Results on real-world datasets

As the optimal partition algorithm OPTIMAL is not scalable for real datasets, we present comparative results of KGAPPROX with baselines KGOPTDP and KGOPTDS. The KGOPTDP algorithm performs exact dynamic programming, but uses an approximate incremental algorithm for the densest subgraph search (the incremental framework by Epasto et al. [19]). Vice versa, KGOPTDS performs approximate dynamic programming while calculating the densest subgraph optimally for each candidate interval (by Goldberg's algorithm [27]). Note that KGOPTDP has $2(1+\epsilon_{\text{DS}})^2$ approximation guarantee and KGOPTDS has $(1+\epsilon_{\text{DP}})$ approximation guarantee. However, even these non-optimal baselines are quite slow in practice and we use a subset of 1 000 interactions of *Students* and *Enron* datasets for comparative reporting.

To ensure fairness, we report the total density of the optimal densest subgraphs in the intervals returned by the algorithms.

In Table 1, we report the density of the solutions reported by KGAPPROX, KGOPTDP, and KGOPTDS, and Table 2 shows their running time. We experiment with different parameters for

Table 1 Comparison of KGAPPROX with KGOPTDP and KGOPTDS baselines: total community density

Dataset	Community density						
	ϵ_{DP}						
		KGAPPROX	0.01	0.1	1	2	KGOPTDS
Students 1000	ϵ_{DS}	0.01	4.24	4.24	4.24	4.24	6.30
		0.1	4.24	4.24	4.24	4.24	6.22
		1	3.82	3.82	3.82	3.82	5.76
		2	3.82	3.82	3.82	3.82	5.61
		KGOPTDP	5.73	5.73	3.82	3.82	
Enron 1000	ϵ_{DS}	0.01	10.4	10.4	10.0	10.5	11.3
		0.1	10.3	10.4	10.0	10.3	11.0
		1	9.54	9.54	8.80	9.83	11.0
		2	7.34	7.34	7.34	7.34	10.8
		KGOPTDP	10.5	11.0	10.4	8.90	

the approximate densest subgraph search (ϵ_{DS}) and for approximate dynamic programming (ϵ_{DP}).

For both datasets, the best solution (i.e., the solution with the highest value of the profit function of Problem 1) was found by KGOPTDS. This is expected as this algorithm has the best approximation factor. The solution cost decreases as ϵ_{DP} increases. On the other hand, KGOPTDS has the largest running time, which decreases with increasing ϵ_{DP}, but even with the largest parameter value ($\epsilon_{DP} = 2$) KGOPTDS takes about an hour.

The KGOPTDP algorithm typically finds the second-best solution; however it only marginally outperforms KGAPPROX (e.g., $\epsilon_{DS} = 0.1$), while requiring up to several orders of magnitude of higher computational time. Naturally, the quality of the solution degrades with increasing ϵ_{DS}.

The solution quality degrades with increasing the approximation parameters for all algorithms. However, the degradation is not as dramatic as the worst-case bound suggests, while using such an approximation parameter offers significant speedup. KGAPPROX provides the fastest estimates of a good quality for a wide range of approximation parameters. Note that KGAPPROX is more sensitive to the changes in the quality of the densest subgraph search regulated by ϵ_{DS}.

5.6 Running time and scalability

Figure 5 shows running time of KGAPPROX as a function of the approximation parameters ϵ_{DS} and ϵ_{DP}. The figure confirms the theory, that is, ϵ_{DS} has significant impact on the running time, while the algorithm scales very well with ϵ_{DP}.

We demonstrate scalability in Fig. 6, plotting the running time for increasing number of interactions, for *Facebook* and *Twitter* datasets. Recall that the theoretical running time is $\mathcal{O}(k^2 m \log n)$, where n is the number of nodes and m the number of interactions. In practice, the running time grows fast for the first thousand interactions and then saturates to linear dependence. This happens because in the beginning of the network history the number of nodes grows fast. In addition, new, denser than previously seen, subgraphs are more likely to

Table 2 Comparison of kGapprox with kGoptDP and kGoptDS baselines: total community density: running time

Dataset	Running time (sec)						
	ϵ_{DP}						
		kGapprox	0.01	0.1	1	2	kGoptDS
Students 1000	ϵ_{DS}	0.01	0.62	0.62	0.63	0.64	23678
		0.1	0.23	0.23	0.24	0.23	8952
		1	0.13	0.26	0.13	0.13	3394
		2	0.36	0.20	0.20	0.36	3769
		kGoptDP	162	43.5	29.5	13.23	
Enron 1000	ϵ_{DS}	0.01	56.4	55.5	42.3	31.8	25788
		0.1	3.02	2.85	2.07	1.70	16070
		1	0.43	0.44	0.29	0.28	7834
		2	0.22	0.22	0.23	0.23	3469
		kGoptDP	1654	61.15	17.82	6.07	

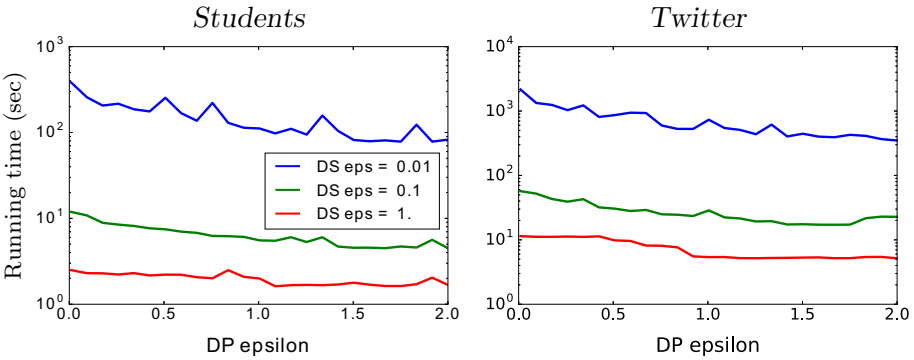

Fig. 5 Effect of different approximation parameters in kGapprox. $k = 20$

occur. Thus, the approximate densest subgraph subroutine has to be computed more often. Furthermore, the number of intervals k contributes to running time as expected.

Figure 7 shows how the cost of the solution changes as the network evolves. Setting larger k results in larger total density. However, the relative change of the solution values is approximately the same for all k: As the number of time stamps goes from 100 to 100000, the total density increases about 2.5 times for *Facebook* dataset and 3.5 times for *Twitter* dataset. This means that while different k lead to technically different segmentations, they capture the rate of network evolution.

Naturally, setting larger k results in discovering subgraphs of smaller individual density, as it follows from Fig. 8. However, the relative difference between the mean density for different k is typically less than the relative difference between the values of k itself. This means that the algorithm tends not to split intervals of dense subgraphs to achieve a better total density, but rather discovers new dense subgraph intervals as k increases.

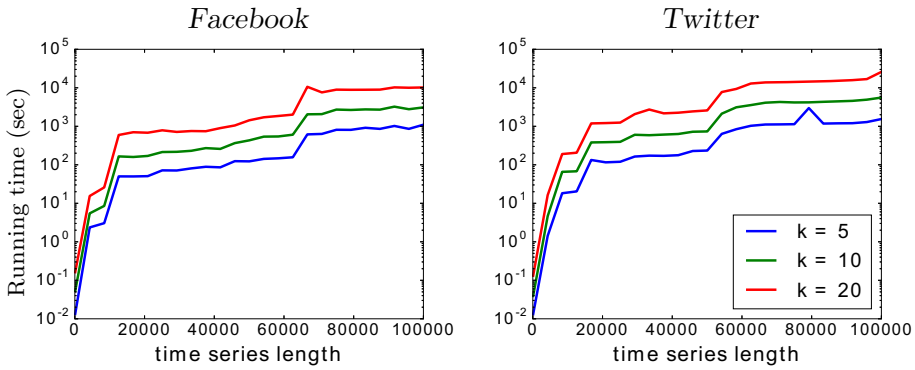

Fig. 6 Scalability testing with $\epsilon_{\text{DS}} = \epsilon_{\text{DP}} = 0.1$

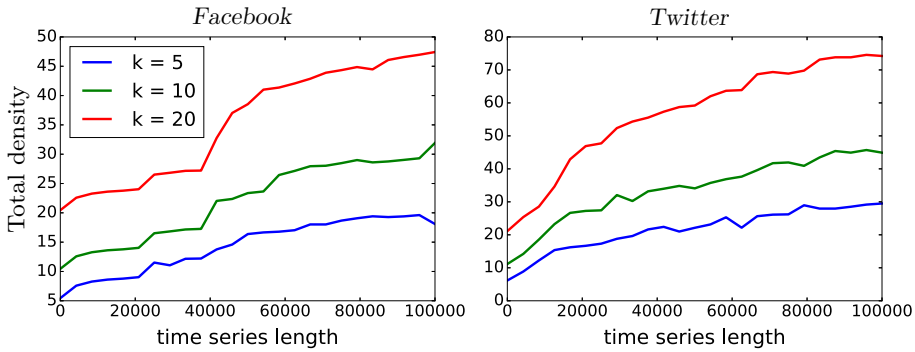

Fig. 7 Total density of the solution subgraphs for different values of k and different lengths of the time series ($\epsilon_{\text{DS}} = \epsilon_{\text{DP}} = 0.1$)

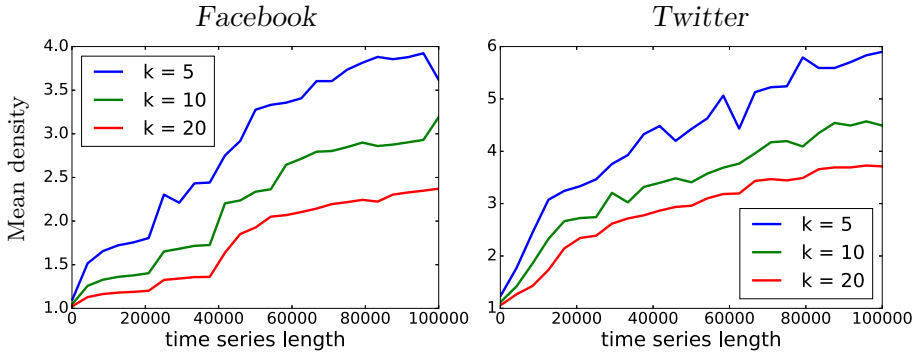

Fig. 8 Mean density of the solution subgraphs for different values of k and different lengths of the time series ($\epsilon_{\text{DS}} = \epsilon_{\text{DP}} = 0.1$)

5.7 Subgraphs with larger node coverage—static graphs

Next, we evaluate StaticGreedy. To measure coverage, we simply count the number of distinct nodes in the output subgraphs. We use the 10K first interactions of *Students* dataset,

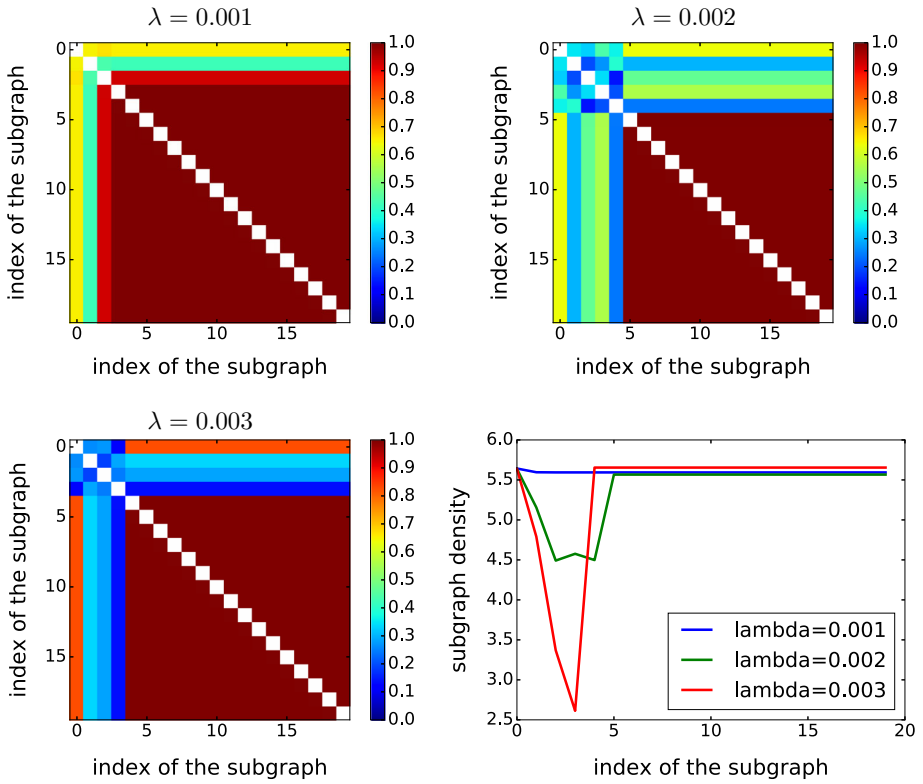

Fig. 9 Pairwise similarities (three heatmap plots on the left) and densities (right plot) of subgraphs returned by StaticGreedy

set $k = 20$, and test different values of λ. Figure 9 shows the density and the pairwise Jaccard similarity of the node sets of the retrieved subgraphs. The subgraphs are shown in the order they are discovered. Smaller values of λ give larger density, and larger values of λ give more cover. We observe that, for all values of λ, in the beginning StaticGreedy returns diverse and dense subgraphs, but soon after it starts outputting graphs, which have been already selected to the solution on the previous iterations. We speculate that the algorithm finds all dense subgraphs that exist in the dataset. Regarding setting λ, we observe that $\lambda = 0.002$ offers a good trade-off in finding subgraphs of high density and moderate overlap.

5.8 Subgraphs with larger node coverage—dynamic graphs

Finally, we evaluate the performance of kGCvr algorithm. We vary the parameter λ and compare different characteristics of the solution, with the solution returned by kGApprox. For different values of λ, Table 3 shows average density and total number of covered nodes, and Table 4 shows average size of the subgraphs and average pairwise Jaccard similarity. Although kGCvr does not have an approximation guarantee, for small values of λ it finds subgraphs of the density close to kGApprox. Similarly to the static case, λ provides an efficient trade-off between density and coverage.

Table 3 Total density and total cover size of kGCvr's outputs with $k = 5$ and $\epsilon_{DS} = \epsilon_{DP} = 0.1$

Dataset	λ	Density		Cover	
		kGCvr	kGApprox	kGCvr	kGApprox
Students	1e−6	10.690	11.151	136	130
	1e−5	7.0869	11.151	813	130
	1e−4	5.0273	11.151	889	130
Enron	1e−6	19.995	19.871	38	37
	1e−5	19.962	19.871	40	37
	1e−4	6.5684	19.871	1144	37
Facebook	1e−8	5.3714	5.3933	83	120
	1e−7	4.2749	5.3933	3470	120
	1e−6	3.2673	5.3933	4100	120
Twitter	1e−7	9.9970	10.138	128	152
	1e−6	6.5500	10.138	3808	152
	1e−5	3.5389	10.138	4604	152

Table 4 Average subgraph size and average Jaccard similarity between the subgraphs in the output of kGCvr with $k = 5$ and $\epsilon_{DS} = \epsilon_{DP} = 0.1$

Dataset	λ	Size		JSim	
		kGCvr	kGApprox	kGCvr	kGApprox
Students	1e−6	48.75	37.6	0.1449	0.0951
	1e−5	261.0	37.6	0.0788	0.095
	1e−4	286.0	37.6	0.0910	0.0951
Enron	1e−6	16.0	16.2	0.3619	0.3851
	1e−5	17.0	16.2	0.3660	0.3851
	1e−4	288.8	16.2	0.0808	0.3851
Facebook	1e−8	22.75	27.6	0.0185	0.0163
	1e−7	882.0	27.6	0.0027	0.0163
	1e−6	1228.75	27.6	0.0335	0.0163
Twitter	1e−7	44.25	54.0	0.1590	0.1673
	1e−6	1061.75	54.0	0.0837	0.1673
	1e−5	1379.0	54.0	0.0773	0.1673

5.9 Parameter selection

Both problem formulations, k- Densest- Episodes and k- Densest- Episodes- EC, follow the classic sequence segmentation problem setting [10] and take as input the number of segments (k) in the timeline partition. It is primarily assumed that the value of k can be specified by prior knowledge and user expectation. In the case of problem formulations k- Densest- Episodes and k- Densest- Episodes- EC, we can show ("Appendix A") that the total profit is a strictly increasing function of the number of segments and reaches its maximum when k is equal to the number of intervals. Thus, the value of k cannot be guided by the optimal value. Furthermore, it is hard to assess the quality of the subgraphs in the segmentation: Larger

intervals with denser subgraphs correspond to larger events, while splitting an interval in favor of less dense subgraphs corresponds to sub-events. Duplicating events in the neighboring intervals can also lead to different sub-segmentation, when k increases; thus, we cannot recommend to decrease k if duplicates occur. However, we do not view that uncertainty with respect to the choice of k as a weakness of the approach: It allows the user to explore the data at different granularity levels and possibly observe a hierarchy of events.

The problem formulations kGApprox and kGCvr require the approximation parameters ϵ_{DP} and ϵ_{DS}. As we discussed in the section about the performance of kGApprox, kGoptDP, and kGoptDS (Table 1), by design our approximation algorithms are more sensitive to the changes in the quality of the densest subgraph search. The parameter ϵ_{DS} affects the calculation of profits of the intervals, and these values are used to guide the dynamic programming algorithm, while loose values of these approximation parameters are likely to misguide it. As it follows from the scalability results (Fig. 5), the algorithms scale better with the change of ϵ_{DS} rather than ϵ_{DS}. However, both parameters contribute equally to the solution quality guarantee $2(1+\epsilon_{DS})(1+\epsilon_{DP})$ of Problems k- Densest- Episodes and the order of the approximation factor depended on the largest of ϵ_{DS} and ϵ_{DP}. Thus, it is not guaranteed (and not fully supported by empirical results) that reducing only ϵ_{DS} will lead to better results faster. As a rule of thumb in most of our experiments, we use $\epsilon_{DS} = \epsilon_{DP} = 0.1$, which gives a satisfactory guarantee of 2.42 and is sufficiently fast. We use the same parameters for the experiments with kGCvr.

The last parameter to discuss is the parameter λ in problem k- Densest- Episodes- EC, which controls the node coverage in the solution. The sensitivity and the range of meaningful values of this parameter depend non-trivially on the topological and temporal properties of the network. To select a good value for λ, one could try sampling different values and plot the density of the resulting subgraphs, similarly to Fig. 9. Then, one can choose a value for λ, which provides a good trade-off between diversity and density: Too small value of λ may lead to dense but repeating structures, and too large value may yield too large and not dense-enough subgraphs.

6 Case study

We present a case study using graphs of co-occurring hashtags from Twitter messages in the Helsinki region. We create two subsets of *Twitter#* dataset: one covering all tweets in November 2013 and another in December 2013. November dataset consists of 4758 interactions, 917 nodes, and the corresponding static graph has average degree density 3.546. December dataset has 5559 interactions, 1039 nodes, and the density is 3.290.

Figures 10 and 11 show the dense subgraphs discovered by the kGApprox algorithm on these datasets, with $k = 4$ and $\epsilon_{DS} = \epsilon_{DP} = 0.1$.

For the November dataset, kGApprox creates a small 1-day interval in the beginning and then splits the rest time almost evenly. This first interval includes the nodes `movember`, `liiga`, `halloween`, and `digiexpo`, which cover a broad range of global (e.g., movember and Halloween) and local events (e.g., game industry event DigiExpo and Finnish ice hockey league). The next interval is represented by a large variety of well-connected tags related to `mtv` and media, corresponding to the MTV Europe Music Awards 13 on November 10. There are also other ice hockey-related tags, e.g., `leijonat` and Father's Day tags, e.g., `isänpäivä`, which was on November 13. The third interval is mostly represented by Slush-related tags; Slush is the annual large startup and tech event in Helsinki. The last interval is completely dedicated to ice hockey with many team names.

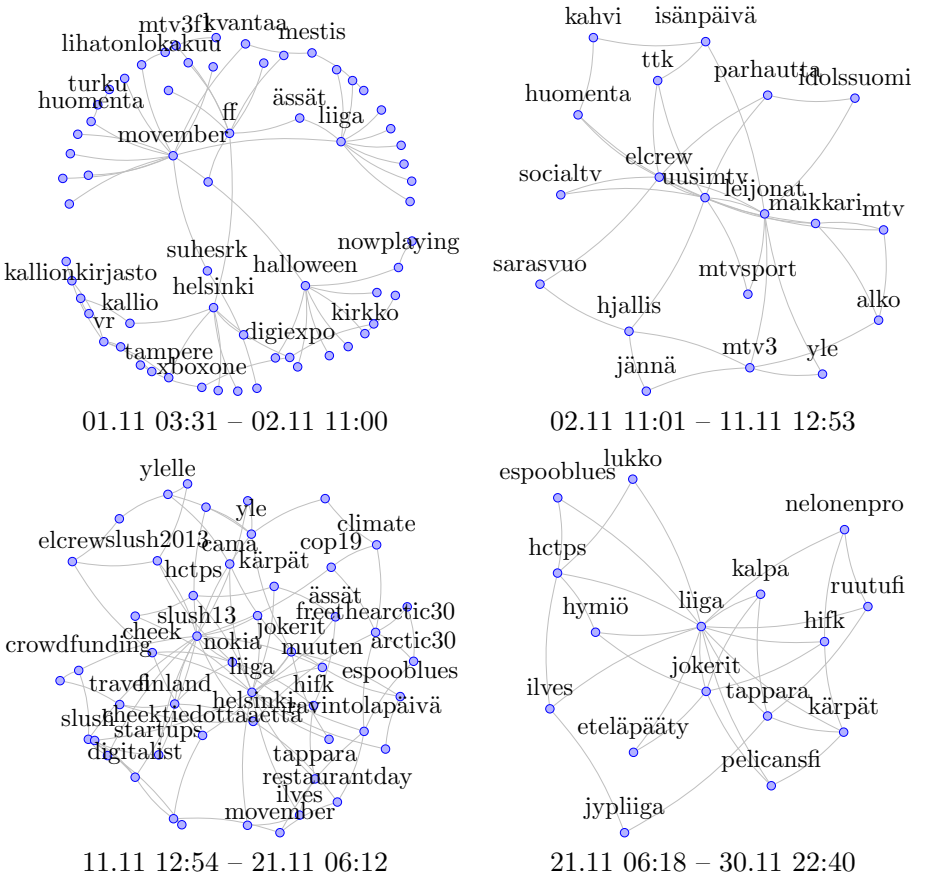

Fig. 10 Subgraphs, discovered in the network of Twitter hashtags *Twitter#* from November 2013 KGAPPROX algorithm with $k = 4$, $\epsilon_{DS} = \epsilon_{DP} = 0.1$

There are three major public holidays in December: Finland's Independence Day on December 6, Christmas on December 25, and New Year's Eve on December 31. KGAPPROX allocates one interval for Christmas and New Year from December 21 to 31. Ice hockey is also represented in this interval, as well as in the third interval. Remarkably, the Independence Day holiday is split into two intervals. The first one is from December 1 to December 6, 3:30pm, and the corresponding graph has two clusters: the first one contains general holidays-related tags and the second one is focused on Independence Day President's reception. (Itsenäisyyspäivän vastaanotto or colloquially Linnan juhlat/Slotts balen). This is a large event that starts on December 6, 6pm, is broadcasted live, and is discussed in media for the following days. The second interval for December 6–9 is a truthful representation of this event.

To demonstrate the qualitative performance of KGAPPROX for different parameters, we consider three parameters settings: $case_1$: $\epsilon_{DS} = 0.1, \epsilon_{DP} = 0.1$; $case_2$: $\epsilon_{DS} = 0.01, \epsilon_{DP} = 0.1$; and $case_3$: $\epsilon_{DS} = 0.1$, $\epsilon_{DP} = 0.01$. Table 5 shows the characteristics of the solution graphs (H_1, H_2, H_3, H_4) discovered in the different settings.

The first two rows show the average degree density and the number of nodes (size) of each graph. Rows 3 and 4 compare an ith graph in one solution (i.e., in one parameters

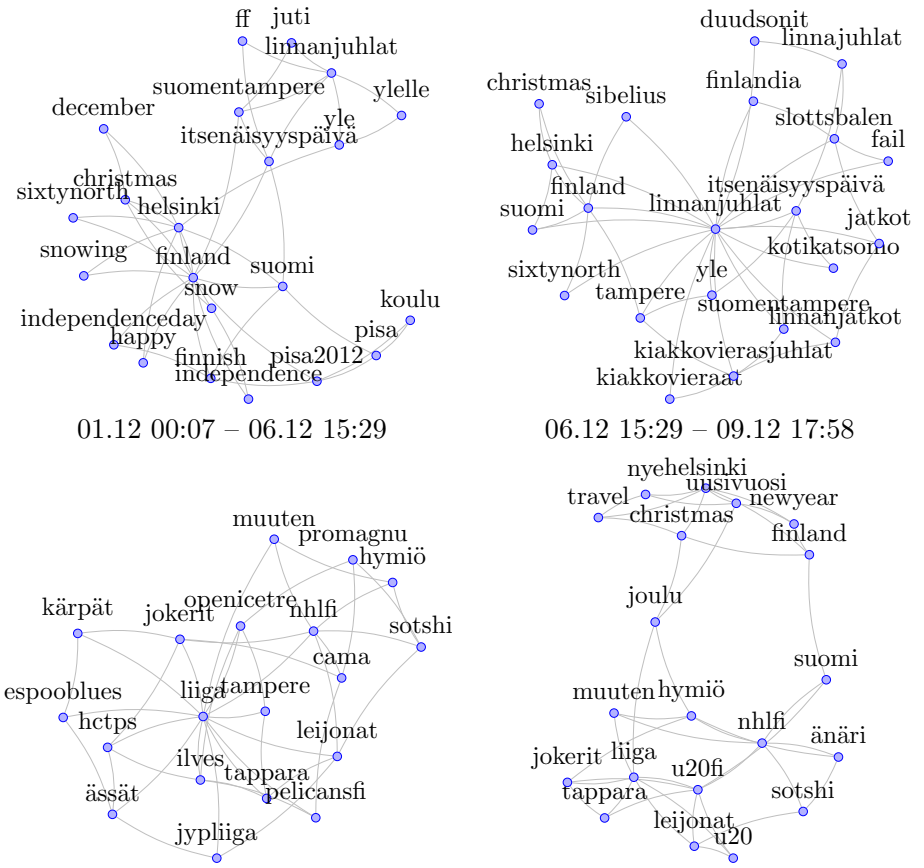

Fig. 11 Subgraphs, discovered in the network of Twitter hashtags *Twitter#* from December 2013 by KGAPPROX algorithm with $k = 4$, $\epsilon_{DS} = \epsilon_{DP} = 0.1$

setting) with the ith graph in other solutions (i.e., in other parameters setting). We report the average overlap in nodes and average Jaccard similarity of node sets. Larger overlap and larger Jaccard similarity values provide evidence that the algorithm outputs similar i-th episode graphs for different settings. For the November datasets, the first two episodes are identical for all settings. Episodes 3 and 4 are similar for cases $case_1$ and $case_3$, but different for $case_2$: As it is discussed before, the change in the densest subgraph search contributes to the change in the solution. There is a similar trend for the December dataset, although the similarity values are typically lower.

Rows 5 and 6 present the similarities between the graphs in *one* solution. We compare an i-th graph in a solution to all other episode graphs in that solution. We report an average overlap in nodes and average Jaccard similarity of node sets. Lower overlap and smaller Jaccard similarity values indicate that the graphs in the solution differ. All similarity values for both datasets are quite low. Although the average overlap in nodes can be as high as 7.333, such an overlap is not prominent when the sizes of the graphs are taken into consideration, as it is shown by the Jaccard similarity metric.

Table 5 Characteristics of the episode graphs H_i discovered for different parameters of ϵ_{DS} and ϵ_{DP} in the case-study dataset

Metric	November				December			
	H_i	$case_1$	$case_2$	$case_3$	H_i	$case_1$	$case_2$	$case_3$
Density	H_1	1.8	1.8	1.8	H_1	3.647	4.857	3.647
	H_2	3.487	3.487	3.487	H_2	3.778	3.867	3.778
	H_3	3.563	5.091	3.563	H_3	4.4	3.6	2.667
	H_4	4.0	1.333	4.0	H_4	4.1	3.0	5.302
Size	H_1	10	10	10	H_1	17	14	17
	H_2	39	39	39	H_2	19	30	9
	H_3	32	22	32	H_3	35	10	6
	H_4	9	3	9	H_4	20	6	43
Avg. overlap (across the solutions)	H_1	10	10	10	H_1	14.5	12.0	14.5
	H_2	39	39	39	H_2	6.0	3.0	6.0
	H_3	22	12	22	H_3	6.5	4.5	3.0
	H_4	4.5	0	4.5	H_4	12.0	5.5	12.5
Avg. Jac. similarity (across the solutions)	H_1	1.0	1.0	1.0	H_1	0.816	0.632	0.816
	H_2	1.0	1.0	1.0	H_2	0.541	0.083	0.542
	H_3	0.643	0.285	0.643	H_3	0.178	0.141	0.103
	H_4	0.5	0.0	0.5	H_4	0.335	0.189	0.286
Avg. overlap (inside the solution)	H_1	3.667	3.0	3.667	H_1	5.333	1.333	4.0
	H_2	4.333	4.333	4.333	H_2	4.667	5.333	3.667
	H_3	5.0	4.667	5.0	H_3	7.333	3.667	1.333
	H_4	3.0	0	3.0	H_4	6.667	2.333	4.333
Avg. Jac. similarity (inside the solution)	H_1	0.127	0.091	0.127	H_1	0.199	0.033	0.153
	H_2	0.086	0.087	0.081	H_2	0.195	0.158	0.151
	H_3	0.108	0.119	0.108	H_3	0.172	0.160	0.030
	H_4	0.113	0.0	0.113	H_4	0.182	0.119	0.088

$case_1$: $\epsilon_{DS} = 0.1, \epsilon_{DP} = 0.1$; $case_2$: $\epsilon_{DS} = 0.01, \epsilon_{DP} = 0.1$; $case_3$: $\epsilon_{DS} = 0.1, \epsilon_{DP} = 0.01$

We can conclude that for all parameters, the solution for the case study consists of diverse graphs. However, changing the accuracy of the densest subgraphs search may lead to differences in the output episodes graphs.

7 Related work

Partitioning a graph in dense subgraphs is a well-established problem. Many of the existing works adopt as density definition the average-degree notion [2,23,33,50]. The densest subgraph, under this definition, can be found in polynomial time [27]. Moreover, there is a 2-approximation greedy algorithm by Charikar [15] and Asahiro et al. [4], which runs in linear time of the graph size. Many recent works develop methods to maintain the average-degree densest subgraph in a streaming scenario [14,19,20,38,39]. Alternative density definitions,

such as variants of quasi-clique, are often hard to approximate or solve by efficient heuristics due to connections to **NP**-complete Maximum Clique problem [1,37,49].

A line of work focuses on dynamic graphs, which model node/edge additions/deletions. Different aspects of network evolution, including evolution of dense groups, were studied in this setting [6,11,34,41]. However, here we use the interaction network model, which is different to dynamic graphs, as it captures the instantaneous interactions between nodes.

Another classic approach to model temporal graphs is to consider graph snapshots, find structures in each snapshot separately (or by incorporating information from previous snapshots), and then summarize historical behavior of the discovered structures [5,12,28,36,40]. These approaches usually focus on the temporal coherence of the dense structures discovered in the snapshots and assume that the snapshots are given. In this work, we aggregate instantaneous interaction into timeline partitions of arbitrary lengths.

To the best our knowledge, the following works are better aligned with our approach. A work of Rozenshtein et al. [44] considers a problem of finding the densest subgraph in a temporal network. However, first, they do not aim at creating a temporal partitioning. Second, they are interested in finding a single dense subgraph whose edges occur in k short time intervals. On the contrary, in this work we search for an interval partitioning and consider only graphs that are span continuous intervals. Other close works are by Jethava and Beerenwinkel [31] and Semertzidis et al. [46]. However, these works consider a set of snapshots and search for a single heavy subgraph induced by one or several intervals. The work of Semertzidis et al. [46] explores different formulations for the *persistent heavy subgraph problem*, including maximum average density, while Jethava and Beerenwinkel [31] focus solely on maximum average density.

8 Conclusions

In this work, we consider the problem of finding a sequence of dense subgraphs in a temporal network. We search for a partition of the network timeline into k non-overlapping intervals, such that the intervals span subgraphs with maximum total density. To provide a fast solution for this problem, we adapt recent work on dynamic densest subgraph and approximate dynamic programming. In order to ensure that the episodes we discover consist of a diverse set of nodes, we adjust the problem formulation to encourage coverage of a larger set of nodes. While the modified problem is **NP**-hard, we provide a greedy heuristic, which performs well on empirical tests.

The problems of temporal event detection and timeline segmentation can be formulated in various ways depending on the type of structures that are considered to be interesting. Here, we propose segmentation with respect to maximizing subgraph density. The intuition is that those dense subgraphs provide a sequence of interesting events that occur in the lifetime of the temporal network. However, other notions of interesting structures, such as frequency of the subgraphs, or statistical non-randomness of the subgraphs, can be considered for future work. In addition, it could be meaningful to allow more than one structure per interval. Another possible extension is to consider overlapping intervals instead of a segmentation.

Acknowledgements Open access funding provided by Aalto University. Part of this work was done while the first author was visiting ISI Foundation. This work was partially supported by three Academy of Finland Projects (286211, 313927, and 317085) and the EC H2020 RIA Project "SoBigData" (654024). We thank the anonymous reviewers for their valuable comments.

A Supporting proofs

For the ease of notations, let us write $[t]$ for the one-time stamp interval $[t, t]$.

Proposition 8 *The total profit of optimal solution of* k- DENSEST- EPISODES *is a strictly increasing function of the number of segments* k *and reaches its maximum when* k *is equal to the number of intervals.*

Proof Given a temporal graph $G = (V, \mathcal{T}, E)$ with m time stamps,

let $S^k = \left\{(I_\ell^k, H_\ell^k)\right\}$, for $\ell = 1, \ldots, k$ and be the solution for k- DENSEST- EPISODES for some $k < m$. Let $S^{k'} = \left\{(I_\ell^{k'}, H_\ell^{k'})\right\}$, for $\ell = 1, \ldots, k'$ be the solution for k- DENSEST- EPISODES for some other $k' = k + 1$.

We will show that the profit value for S^k is less than the profit value for $S^{k'}$: $\sum_{\ell=1}^{k} d(H_\ell^k) < \sum_{\ell=1}^{k'} d(H_\ell^{k'})$.

Denote the profit of a solution S as $Pr(S)$.

Fix some $\ell \in [1, \ldots, k]$ so that the corresponding episode $((I_\ell^k, H_\ell^k))$ of S^k has the interval I_ℓ^k with more than one time stamp. By the problem definition, H_ℓ^k is the densest subgraph of the interval I_ℓ^k. Let (E^*, V^*) be edges and nodes of H_ℓ^k. Now consider an arbitrary split of I_ℓ^k into L and R non-empty intervals and construct a new sequence of episodes S' with $k + 1$ episode, which is the same as S^k except for the split episode I_ℓ^k.

Let $E^*[L]$ be the subset of E^*, which appear in L, and $E^*[R]$ be the subset of E^*, which appear in R. Similarly, define $V^*[L]$ and $V^*[R]$.

Now the following is true for the density of the densest subgraph in L:

$d^*(G[L]) \geq 2\frac{|E^*[L]|}{|V^*[L]|} \geq 2\frac{|E^*[L]|}{|V^*|}$.

The same inequality can be written for $d^*(G[R])$.

Also $|E^*[L]| + |E^*[R]| \geq |E^*|$.

Thus,

$d^*(G[L]) + d^*(G[R]) \geq 4\frac{|E^*[L]|+|E^*[R]|}{|V^*|} \geq 4\frac{|E^*|}{|V^*|} = 2d^*(G[I_\ell^k]) > d^*(G[I_\ell^k])$.

This leads to the larger profit of segmentation S': $Pr(S^k) < Pr(S')$.

Splitting the interval I_ℓ^k into L and R gives a $k + 1$ segmentation, which profit is by optimality of $S^{k'}$ should be not larger than the profit of $S^{k'}$: $Pr(S') \leq Pr(S^{k'})$.

Thus, $Pr(S^k) < Pr(^{k'})$ and we conclude the proof. □

Similar statement can be proven for k- DENSEST- EPISODES- EC in the same way: As cover is a nonnegative and non-decreasing function of the subgraphs (Proposition 6), splitting an episode is still always beneficial.

References

1. Alvarez-Hamelin J I, Dall'Asta L, Barrat A, Vespignani A (2006) Large scale networks fingerprinting and visualization using the k-core decomposition. In: NIPS
2. Andersen R, Chellapilla K (2009) Finding dense subgraphs with size bounds. In: WAW, pp 25–37

3. Angel A, Sarkas N, Koudas N, Srivastava D (2012) Dense subgraph maintenance under streaming edge weight updates for real-time story identification. PLVDB 5(6):574–585
4. Asahiro Y, Iwama K, Tamaki H, Tokuyama T (2000) Greedily finding a dense subgraph. J Algorithms 34(2):203–221
5. Asur S, Parthasarathy S, Ucar D (2009) An event-based framework for characterizing the evolutionary behavior of interaction graphs. TKDD 3(4):16
6. Backstrom L, Huttenlocher D, Kleinberg J, Lan X (2006) Group formation in large social networks: membership, growth, and evolution. In: KDD, pp 44–54
7. Balalau OD, Bonchi F, Chan T, Gullo F, Sozio M (2015) Finding subgraphs with maximum total density and limited overlap. In: WSDM, pp 379–388
8. Balalau O D, Castillo C, Sozio M (2018) Evidense: a graph-based method for finding unique high-impact events with succinct keyword-based descriptions. In: Proceedings of the twelfth international conference on web and social media, ICWSM, pp 560–563
9. Bellman R (2013) Dynamic programming, Courier Corporation
10. Bellman R, Kotkin B (1962) On the approximation of curves by line segments using dynamic programming. II, Technical report, RAND CORP SANTA MONICA CALIF
11. Berlingerio M, Bonchi F, Bringmann B, Gionis A (2009) Mining graph evolution rules. In: ECML PKDD, pp 115–130
12. Berlingerio M, Pinelli F, Calabrese F (2013) Abacus: frequent pattern mining-based community discovery in multidimensional networks. DMKD 27(3):294–320
13. Beutel A, Xu W, Guruswami V, Palow C, Faloutsos C (2013) Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In: WWW, pp 119–130
14. Bhattacharya S, Henzinger M, Nanongkai D, Tsourakakis C (2015) Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In: STOC, pp 173–182
15. Charikar M (2000) Greedy approximation algorithms for finding dense components in a graph. In: APPROX, pp 84–95
16. Chen J, Saad Y (2012) Dense subgraph extraction with application to community detection. TKDE 24(7):1216–1230
17. Danisch M, Chan T H, Sozio M (2017) Large scale density-friendly graph decomposition via convex programming. In: Proceedings of the 26th international conference on World Wide Web, WWW 2017
18. DiTursi D, Ghosh G, Bogdanov P (2017) Local community detection in dynamic networks. arXiv:1709.04033
19. Epasto A, Lattanzi S, Sozio M (2015) Efficient densest subgraph computation in evolving graphs. In: WWW, pp 300–310
20. Esfandiari H, Hajiaghayi M, Woodruff D (2015) Applications of uniform sampling: Densest subgraph and beyond. arXiv:1506.04505
21. Feder T, Motwani R (1995) Clique partitions, graph compression and speeding-up algorithms. JCSS 51(2):261–272
22. Fratkin E, Naughton BT, Brutlag DL, Batzoglou S (2006) Motifcut: regulatory motifs finding with maximum density subgraphs. Bioinformatics 22(14):e150–e157
23. Galbrun E, Gionis A, Tatti N (2014) Overlapping community detection in labeled graphs. DMKD 28(5–6):1586–1610
24. Galbrun E, Gionis A, Tatti N (2016) Top-k overlapping densest subgraphs. DMKD 30(5):1134–1165
25. Gallo G, Grigoriadis MD, Tarjan RE (1989) A fast parametric maximum flow algorithm and applications. SIAM J. Comput. 18:30–55
26. Gibson D, Kumar R, Tomkins A (2005) Discovering large dense subgraphs in massive graphs. In: PVLDB, pp 721–732
27. Goldberg A V (1984) Finding a maximum density subgraph, University of California Berkeley
28. Greene D, Doyle D, Cunningham P (2010) Tracking the evolution of communities in dynamic social networks. In: ASONAM
29. Guha S, Koudas N, Shim K (2001) Data-streams and histograms. In: STOC, pp 471–475
30. Hernández C, Navarro G (2012) Compressed representation of web and social networks via dense subgraphs. In: SIGIR, pp 264–276
31. Jethava V, Beerenwinkel N (2015) Finding dense subgraphs in relational graphs. In: ECML PKDD, pp 641–654
32. Karande C, Chellapilla K, Andersen R (2009) Speeding up algorithms on compressed web graphs. Internet Math 6:373–398
33. Khuller S, Saha B (2009) On finding dense subgraphs. In: ICALP
34. Li R-H, Yu JX, Mao R (2014) Efficient core maintenance in large dynamic graphs. TKDE 26(10):2453–2465

35. Lin H, Bilmes J (2011) A class of submodular functions for document summarization. In: ACL, pp 510–520
36. Lin Y-R, Chi Y, Zhu S, Sundaram H, Tseng B L (2008) Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In: WWW, pp 685–694
37. Makino K, Uno T (2004) New algorithms for enumerating all maximal cliques. In: SWAT, pp 260–272
38. McGregor A, Tench D, Vorotnikova S, Vu HT (2015) Densest subgraph in dynamic graph streams. In: MFCS. Springer, Berlin
39. Mitzenmacher M, Pachocki J, Peng R, Tsourakakis C, Xu SC (2015) Scalable large near-clique detection in large-scale networks via sampling. In: KDD, pp 815–824
40. Mucha PJ, Richardson T, Macon K, Porter MA, Onnela J-P (2010) Community structure in time-dependent, multiscale, and multiplex networks. Science 328(5980):876–878
41. Myers SA, Leskovec J (2014) The bursty dynamics of the twitter information network. In: WWW, pp 913–924
42. Nemhauser G, Wolsey L, Fisher M (1978) An analysis of approximations for maximizing submodular set functions. Math Progr 14(1):265–294
43. Orlin JB (2013) Max flows in o (nm) time, or better. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing
44. Rozenshtein P, Tatti N, Gionis A (2017) Finding dynamic dense subgraphs. TKDD 11(3):27
45. Saha B, Hoch A, Khuller S, Raschid L, Zhang X-N (2010) Dense subgraphs with restrictions and applications to gene annotation graphs. In: RECOMB
46. Semertzidis K, Pitoura E, Terzi E, Tsaparas P (2018) Finding lasting dense subgraphs. Data Mining and Knowledge Discovery
47. Tatti N (2018) Strongly polynomial efficient approximation scheme for segmentation. arXiv:1805.11170
48. Taylor D, Caceres RS, Mucha PJ (2017) Super-resolution community detection for layer-aggregated multilayer networks. Phys Rev X 7(3):031056
49. Tsourakakis C, Bonchi F, Gionis A, Gullo F, Tsiarli M (2013) Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: KDD, pp 104–112
50. Tsourakakis CE (2014) A novel approach to finding near-cliques: the triangle-densest subgraph problem. arXiv:1405.1477
51. Viswanath B, Mislove A, Cha M, Gummadi K (2009) On the evolution of user interaction in facebook, In: WOSN, pp 37–42

Polina Rozenshtein received a M.Sc. degree and an Ph.D. degree from Aalto University, Espoo, Finland, in 2014 and 2018. She is currently a Senior Data Scientist at Nordea Data Science Lab, Helsinki, Finland. Prior to that, she was a postdoctoral researcher in Data Mining Group of Computer Science Department at Aalto University. Her research interests include data mining, combinatorial optimization, dynamic graph mining, social networks analysis, computational social science, and data analysis for social good.

Francesco Bonchi is Deputy Director at the ISI Foundation, Turin, Italy, with responsibility over the Industrial Research area. At ISI Foundation, he is also Research Leader for the "Algorithmic Data Analytics" group. He is also (part-time) Research Director for Big Data & Data Science at Eurecat (Technological Center of Catalunya), Barcelona. Previously, he was Director of Research at Yahoo Labs Barcelona, where he was leading the Web Mining Research group. He has been the General Chair of IEEE DSAA 2018, PC Chair of ECML PKDD'18, ACM HT'17, IEEE ICDM'16, ECML PKDD 2010. He is a member of the Steering Committee of ECML PKDD and IEEE DSAA and associate editor of many journals in the data management and mining area (IEEE TBD, IEEE TKDE, ACM TKDD, ACM TIST, DMKD). More information at http://www.francescobonchi.com/.

Aristides Gionis is a professor in the Department of Computer Science in Aalto University. He is currently a fellow in the ISI foundation, Turin, while he has been a visiting professor in the University of Rome. Previously, he has been a senior research scientist and group leader in Yahoo! Research, Barcelona. He obtained his Ph.D. in 2003 from Stanford University, USA. He is currently serving as an action editor in the Data Management and Knowledge Discovery Journal (DMKD), an associate editor in the ACM Transactions on Knowledge Discovery from Data (TKDD), and an associate editor in the ACM Transactions on the Web (TWEB). He has contributed in several areas of data science, such as algorithmic data analysis, Web mining, social media analysis, data clustering, and privacy-preserving data mining. His current research is funded by the Academy of Finland (Projects Nestor, Agra, AIDA) and the European Commission (Project SoBigData).

Mauro Sozio is currently associate professor in the Department of Computer Science at Telecom ParisTech University in Paris, France. Previously, he held a visiting scientist position at IBM Almaden (USA) and a senior researcher position at the Max-Planck Institute for Informatics (Germany). He received his Ph.D. in computer science from "Sapienza" University of Rome in 2007. He has contributed in several research areas of computer science, such as graph mining and social network analysis, approximation algorithms, and distributed algorithms. He serves or he has served as PC or senior PC member in top venues for data mining, the Web, and databases such as TheWebConf, KDD, PVLDB, ICDM, and others where he has also published more than 20 research papers.

Nikolaj Tatti is an associate professor at University of Helsinki. Previously, he was a senior data scientist at F-Secure, a HIIT research fellow in Aalto University, and an FWO postdoctoral fellow in University of Antwerp. He received his Ph.D. in 2008 from Helsinki University of Technology, Finland. His current research interest is developing and analyzing new data mining methodology with diverse applications. He has published over 60 peer-reviewed papers in top data mining conferences and journals.