SN Computer Science (2021) 2:158
https://doi.org/10.1007/s42979-021-00593-w

ORIGINAL RESEARCH

Dense Sub-networks Discovery in Temporal Networks

Riccardo Dondi[1] · Mohammad Mehdi Hosseinzadeh[1]

Received: 5 March 2021 / Accepted: 13 March 2021 / Published online: 20 March 2021

Abstract
Temporal networks have been successfully applied to analyse dynamics of networks. In this paper we focus on an approach recently introduced to identify dense subgraphs in a temporal network and we present a heuristic, based on the local search technique, for the problem. The experimental results we present on synthetic and real-world datasets show that our heuristic provides mostly better solutions (denser solutions) and that the heuristic is fast (comparable with the fastest method in literature, which is outperformed in terms of quality of the solutions). We present also experimental results of two variants of our method based on two different subroutines to compute a dense subgraph of a given graph.

Keywords Network mining · Temporal networks · Densest Subgraph · Algorithm design · Local search

Introduction

Real-world interactions are usually represented with a network (or a graph), where elements are nodes and relations are edges. In many cases, real-world networks are dynamic, that is new interactions among nodes are observed or existent interactions cease. In order to represent this evolution, the definition of network must be extended to temporal network, so that it includes information about the presence/absence of edges over a temporal time. In particular, a temporal network defines the relations between a set of nodes by considering a discrete number of timestamps and defining for each timestamp which edges (interactions) are active.

Temporal networks have been analyzed to model different kinds of interactions, such as human communication networks, social networks, citation networks, economic networks, brain networks and biological networks [15, 25]. For example, for human communication, consider the data that come from mobile phone operators. Two users that exchange data (via a phone call or a message sent) are nodes of the human communication network and their interaction happens over a time interval [15, 20]. In economic networks, studies of Bitcoin transactions [18] and credit card transactions [28] can be naturally represented as a temporal set of interactions.

Dynamics of human activity (postings, defining new connections, etc.) on social network platforms have been studied in many research communities [10, 26]. As an example, consider interesting events that occur in online social media, such as twitter hashtags, that can be represented as a temporal network [25]. In real-time story identification in online social networks, the aim is to identify emerging stories by looking for dense subgraphs induced by groups of tightly-coupled entities [1].

Analyzing temporal networks provides valuable insights about their framework. Many problems have been studied under temporal dimension of networks, such as community detection [23, 6, 11], frequent subgraph discovery and temporal motifs in time-dependent networks [27, 19], link-based object ranking [8, 13], connectivity for temporal networks [17], and more.

One of the most studied problem in static network mining is the densest subgraph problem. This problem asks for a subgraph of maximum density (or equivalently a subgraph of maximum average-degree density) of an input graph. The densest subgraph problem is particularly relevant for the identification of cohesive subgraphs, which is a fundamental property in graph minining, as such subgraphs are related to relevant parts of networks, such as communities. Although several definitions of cohesive subgraph have been proposed in literature, the dense subgraph problem is

✉ Riccardo Dondi
riccardo.dondi@unibg.it

Mohammad Mehdi Hosseinzadeh
m.hosseinzadeh@unibg.it

[1] Università degli Studi di Bergamo, Via S. Tomaso 40, 24121 Bergamo, Italy

one of the most applied in network mining for this purpose. The densest subgraph problem can be solved in polynomial-time via Goldberg's algorithm [14] and it admits a linear-time greedy algorithm (called Charikar's algorithm) with $\frac{1}{2}$ -approximate factor [2, 5].

In several applications, finding dense subgraphs is relevant to understand the properties of a network. A recent approach called Top-k-Overlapping Densest Subgraphs studied in [12, 7] asks for a collection of top k densest, possibly overlapping, distinct subgraphs and it has been applied also to biological networks [16]. A similar approach has been introduced in [4] to find a set of k subgraphs of maximum density, such that the maximum pairwise Jaccard coefficient of the subgraphs in the solution is bounded. The densest subgraph problem has been applied also to dynamic graphs [9, 21, 22].

Several real-world applications have temporal patterns and thus a natural problem in the analysis of temporal networks is the identification of dense subgraphs. It is interesting to identify a set of intervals that show interesting structures (dense subgraphs). These subgraphs can be considered to be related to interesting episodes, for example a group of users highly interacting in some temporal interval.

The first attempt to consider the discovering of dense subgraphs in a temporal network is the k-Densest-Episodes problem, introduced in [25, 24]. k-Densest-Episodes, given a temporal graph over a certain time domain as input, asks for $k \geq 1$ densest subgraphs that belong to disjoint time intervals. Such a solution can be determined by computing a segmentation of the time domain into not overlapping time intervals and by computing, for each interval of the segmentation, a densest subgraph.

The k-Densest-Episodes problem is polynomial-time solvable [25]. The polynomial-time algorithm computes a segmentation via dynamic programming and a densest subgraph in each time interval of the segmentation via Goldberg's algorithm. While k-Densest-Episodes is polynomial-time solvable, the time complexity of the algorithm is not scalable for large networks [25], hence in [25] other approaches have been considered to tackle the problem. Rozenshtein et al in [25] introduced KGAPPROX, an algorithm based on approximate dynamic programming (ApproxDP, with approximation parameter ϵ_1) and an approximate algorithm for the incremental densest subgraph problem (ApprDens, with an approximation parameter ϵ_2) [9]. KGAPPROX has approximation factor $2(1+\epsilon_1)(1+\epsilon_2)$ and runs in $O\left(\frac{k^2}{\epsilon_1\epsilon_2^2}|\mathcal{T}|m\log^2 n\right)$ time, where k is the number of intervals of a graph with n vertices and m edges, and $\mathcal{T} \subseteq \mathbb{N}$ is a discrete time domain. It follows that decreasing the values of parameters ϵ_1 and ϵ_2 (thus improving the quality of the returned solution), increases the running time of KGAPPROX.

Aiming at designing a fast computational (even if not exact) method to solve k-Densest-Episodes, in this paper we design an efficient heuristic for this problem. Our heuristic, called Local-search Temporal Densest Subgraphs (LSTDS), is based on two techniques. First, an initial segmentation is computed in linear-time based on the number of active edges in timestamps. Then, we apply a local search approach that, at each iteration, aims at possibly increasing the overall density of the k subgraphs by expanding an interval of the segmentation. The computation of a dense subgraph in each interval can be computed either with the Goldberg's algorithm or with the Chiarikar's algorithm.

In "Experimental analysisExperimental analysis" we present experimental results on LSTDS, both on synthetic and real-world networks. First, we compare LSTDS with the algorithm that computes an optimal solution (called OPTIMAL). Due to the time complexity of OPTIMAL, we consider a very small dataset for this comparison. The results show that LSTDS find solutions close to that of OPTIMAL, with significantly lower running time.

Then we generate larger synthetic datasets for further comparisons. We consider how our heuristic is influenced by the algorithm considered for the densest subgraph computation, that is we compare LSTDS using Goldberg's algorithm with LSTDS using Charikar's algorithm. The results show that the solutions returned by LSTDS with Charikar's algorithm have density close to those returned by LSTDS based on Goldberg's algorithm, while, as expected, LSTDS with Charikar's algorithm is significantly faster.

Then, we compare LSTDS with Charikar's algorithm with the KGAPPROX algorihtm given in [25] by considering three different values of ϵ_1 and ϵ_2 and by varying the value of k.

The results on synthetic datasets (see "Synthetic datasets") show that the density of the solutions returned by LSTDS is greater than the solutions returned by the KGAPPROX algorithm for different values of parameters ϵ_1 and ϵ_2. Moreover, LSTDS is significantly faster than KGAPPROX when the parameters ϵ_1 and ϵ_2 are small ($\epsilon_1 = \epsilon_2 = 0.01$ and $\epsilon_1 = \epsilon_2 = 0.1$). For the largest values of ϵ_1 and ϵ_2 considered ($\epsilon_1 = \epsilon_2 = 2$), the running times of KGAPPROX and LSTDS are close, while the solutions returned by LSTDS are significantly denser. We also compare LSTDS and KGAPPROX on the synthetic datasets by considering how well the algorithms find planted communities on 100 independent run. The results show that in many cases LSTDS finds closer solutions to the ground-truth subgraphs with respect to KGAPPROX.

The experiment on real-world datasets (see "Real-world datasets") confirms that LSTDS is efficient and produce high-quality results with respect to KGAPPROX. Following [25], we use four different real-world datasets from social network platforms and emails. On these real-world datasets,

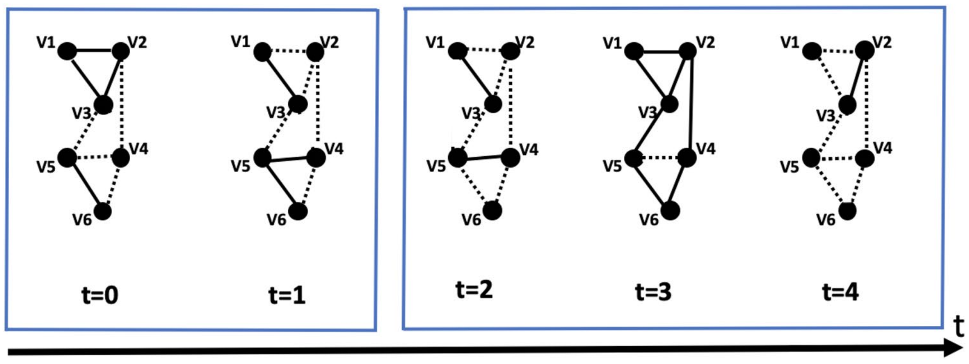

Fig. 1 A temporal network with six nodes and five timestamps [0, 1, 2, 3, 4]. For each timestamp, solid lines represent interactions (edges) that happen at that timestamp, while dotted lines represent interactions (edges) that happen at different timestamps

we vary the number of intervals k and compare the density of the returned solutions and the running time of LSTDS and KGAPPROX. The results show that increasing the number of intervals k has a higher impact on the running time of the KGAPPROX algorithm. In particular, when k is increased to 20, LSTDS is always faster than KGAPPROX.

The paper is organized as follows. In "Introduction", we present some definitions and we give the formal definition of the k-Densest-Episodes problem. In "Method", we introduce the Local-search Temporal Densest Subgraphs algorithm. In "Experimental analysis", we present the experimental results. Finally, we conclude the paper in "Conclusion" with some open problems.

Definition 1 Let $G = (V, \mathcal{T}, E)$ be a temporal graph, an episode is defined as a pair (T, W) where $T \in \mathcal{T}$ is a temporal interval and W is a subgraph of $G[T]$, that is $W = (V_W, E_W)$, where $V_W \subseteq V$ and $E_W \subseteq E[T] \cap (V_W \times V_W)$.

Given a graph $W = (V_W, E_W)$, then the density of W, denoted by dens(W), is defined as follows:

$$\text{dens}(W) = \frac{|E_W|}{|V_W|}.$$

Now, we introduce the problem we will focus in the paper, called k-Densest-Episodes.

Problem 1 k-Densest-Episodes

Input: A temporal graph $G = (V, \mathcal{T}, E)$, a positive number $k \in \mathbb{N}$.

Output: A set S of k episodes, $S = \{(I_j, W_j)\} : 1 \leq j \leq k\}$, where $\{I_j : 1 \leq j \leq k\}$ is a set of disjoint intervals, such that $\sum_{j=1}^{k} \text{dens}(W_j)$ is maximized.

Consider the example of Fig. 1, a temporal network with six nodes $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ and five timestamps [0, 1, 2, 3, 4]. A solution of k-Densest-Episodes with $k = 2$ consists of two episodes, defined over the two disjoint intervals $I_1 = [0, 1]$ and $I_2 = [2, 4]$ (each interval is included in a box). In interval I_1, there are five active interactions/edges $\{\langle v_1, v_2\rangle, \langle v_1, v_3\rangle, \langle v_2, v_3\rangle, \langle v_4, v_5\rangle, \langle v_5, v_6\rangle\}$. The first episode is (I_1, W_1), where W_1 is the subgraph induced by $\{v_1, v_2, v_3\}$. In interval I_2, there are eight active interactions/edges $\{\langle v_1, v_2\rangle, \langle v_1, v_3\rangle, \langle v_2, v_3\rangle, \langle v_3, v_5\rangle, \langle v_4, v_5\rangle, \langle v_4, v_6\rangle, \langle v_5, v_6\rangle\}$. The second episode is (I_2, W_2), where W_2 is the subgraph induced by $\{v_1, v_2, v_3, v_4, v_5, v_6\}$.

Method

In this section, we present our efficient heuristic (LSTDS) for k-Densest-Episodes problem. LSTDS is based on a dense subgraph method (which can be Charikar's algorithm or Goldberg's algorithm) to find a dense subgraph of an active graph.

Given a temporal graph $G = (V, \mathcal{T}, E)$ and parameter $k > 0$, LSTDS first applies a preliminary step and then it iterates local search to possibly improve the density of the computed solution.

We start by describing the preliminary step. In this step, LSTDS starts by computing a partition of the time domain into k disjoint temporal intervals as follows. LSTDS defines a parameter α, where $\alpha = \frac{\sum_{l \in \mathcal{T}} |E_l[t]|}{k}$, that is the overall number of active edges divided by k. Then, the intervals are defined starting from $t = 0$ and moving to $t_{\max}$. LSTDS defines an interval $I_j = [t_{j,1}, t_{j,2}]$, with $j = 1, 2, \ldots, k-1$, where $t_{j,1}$ is the minimum timestamp that does not belong to any defined interval and $t_{j,2}$ is the minimum timestamp such that $|E[I_j]| \geq \alpha$. Once we have computed the $k-1$-th temporal interval $I_{k-1} = [t_{k-1,1}, t_{k-1,2}]$, the k-th interval is defined as $I_k = [t_{k-1,2} + 1, t_{\max}]$.

Then, for each interval, LSTDS applies a dense subgraph method to compute a preliminary solution, that is a set of episodes $S = \{(I_j, W_j) : j = 1, 2, \ldots, k\}$, of total density dens($S$) $= \sum_{j=1}^{k} \text{dens}(W_j)$.

Next, LSTDS uses a local-search strategy in order to possibly improve the density of a computed solution S of k-Densest-Episodes. The episodes of S are defined as marked and unmarked. An unmarked episode of S is an episode that has not been considered as a candidate episode by local search strategy, while the marked episodes of S are those episodes that have already been considered as candidate episodes by local search. When a solution is improved by local search, a new solution S' is created and each episode in it is defined as unmarked.

At each iteration of local search, LSTDS considers a candidate episode (I_c, W_c), such that it is unmarked and W_c has minimum density among the subgraphs belonging to unmarked episodes of S.

Then, in order to improve the density of W_c, LSTDS tries to expand the interval I_c in three possible ways and then computes the density of the new solution, using a dense subgraph method in each modified interval. More precisely, consider parameter $\delta = \frac{|\mathcal{T}|}{4k}$ as a value of expansion. The value of the parameter δ has been chosen in order to provide a significant improvement in the size of the interval associated with the candidate episode, while avoiding to reduce the length of intervals of neighbor episodes to a value close to 0. LSTDS applies the following three different ways for expanding the interval I_c of the candidate episode:

1. Expand from left side, that is $I_{c,1} = [t_i - \delta, t_j]$. This is possible only if the interval I_c doesn't contain the initial timestamp, otherwise LSTDS doesn't apply this expansion.
2. Expand from right side, that is $I_{c,2} = [t_i, t_j + \delta]$. This is possible only if the interval I_c doesn't contain the last timestamp, otherwise LSTDS doesn't apply this expansion.
3. Expand from both sides, that is $I_{c,3} = [t_i - \delta, t_j + \delta]$.

The expansion steps may affect an interval of a neighbor episode, even if this latter episode is already marked. Notice that some expansion steps in some iteration may not be applied, if due to the expansion, the length of an interval of a neighbor episode becomes 0.

Then, for each of three ways of expansion $I_{c,l}$, $1 \leq l \leq 3$, we compute a solution $S_{c,l}$ of k-Densest-Episodes, by applying a dense subgraph method in each interval of $I_{c,l}$. We define $S' = \arg\max(\text{dens}(S_{c,l}))$, $1 \leq l \leq 3$, as the densest set of episodes among the three new solutions.

Then, we have two following cases:

(I) $\text{dens}(S') > \text{dens}(S)$, that is, the value of total density is improved by one of the new solutions. Then, we define $S = S'$ and all the episodes in the updated S are unmarked. Therefore, LSTDS applies the local improvement on the updated set S of episodes.

(II) $\text{dens}(S') \leq \text{dens}(S)$, that is, the value of total density is not improved by one of the new solutions. Then, the candidate episode considered in this iteration is marked and a new iteration of local search is applied.

From a theoretical point of view, LSTDS stops when all episodes of S (possibly computed with some improvements) are marked. Notice that this approach is guaranteed to stop. Indeed, each time a solution S' improves a solution S, it follows that it has higher density. Thus solution S will not be reconsidered by LSTDS and after a finite number of improvements LSTDS will stop.

In practice, in order to speed up the algorithm, we bound the number of iterations $\text{iter}_{\max}$ of LSTDS to k. The value k was chosen so that either the preliminary solution is improved by local search or each episode of the preliminary solution is marked.

Lemma 1 *LSDTS has time complexity $O(|\mathcal{T}| + (k + \text{iter}_{\max})f_{dense})$, where f_{dense} is the time required by a dense subgraph method and $iter_{max}$ is the bound on the number of iterations of LSTDS.*

Proof The preliminary step requires time $O(|\mathcal{T}| + kf_{\text{dense}})$, since LSTDS reads the time domain $\mathcal{T}$ from left to right to compute a segmentation and a dense subgraph method is applied in every interval of the segmentation.

Local search requires $\text{iter}_{\max}$ iterations. In each iteration, an unmarked episode of minimum density is found in constant time by using an ordered list of unmarked episodes. Then, a dense subgraph method is applied at most three times (once for the candidate episode and at most twice for the neighbor episodes). Thus, the overall time complexity of local search is $O(\text{iter}_{\max} f_{\text{dense}})$ and the overall time complexity of LSDTS is $O(|\mathcal{T}| + (k + \text{iter}_{\max})f_{\text{dense}})$. □

Experimental Analysis

In this section, we present the experimental results for our heuristics (LSTDS) on synthetic and real-world datasets. We compare LSTDS with KGAPPROX with parameters $\epsilon_1 = \epsilon_2 = 0.01$, $\epsilon_1 = \epsilon_2 = 0.1$ and $\epsilon_1 = \epsilon_2 = 2$. The values of the parameters have been chosen according to [25]. We implemented LSTDS in Python (notice that also KGAPPROX was implemented in Python). The experiments were run on MacBook-Pro (OS version 10.15.1) with processor 2.9 GHzIntel Core i5 and 8GB 2133 MHz LPDDR3 of RAM, Intel Iris Graphics 550 1536 MB.

Synthetic Datasets

Synthetic temporal networks are generated based on Erdős-Rényi model. Following [25], the temporal graph consists

Table 1 Comparison between LSTDS-Charikar's algorithm, LSTDS-Goldberg's algorithm and OPTIMAL on Synthetic-small

k	LSTDS-Charikar's		LSTDS-Goldberg's		Optimal	
	Time	Density	Time	Density	Time	Density
2	0.02	5.25	0.70	5.25	9.23	5.25
4	0.08	9.34	2.52	9.42	80.64	9.71
6	0.13	12.04	4.45	12.10	137.39	12.72
8	0.23	13.58	6.61	14.11	247.77	14.53
10	0.32	15.66	9.08	16.48	526.66	17.14
12	0.42	16.97	11.48	18.36	664.67	19.62
14	0.57	20.87	14.66	21.44	911.79	22.52

Running time is in seconds

of k planted communities (k planted subgraphs) and a background network. The background network G includes all nodes over the discrete time domain $\mathcal{T}$ varying edge distribution according to average degree of the background graph G (below we specify how density changes in different experiments). The k planted communities G' are defined in non-overlapping intervals and they have the same density.

Three families of synthetic networks, called Synthetic-small, Synthetic1 and Synthetic2, are generated for different purposes, varying time domain, number of nodes/edges, background graph and communities.

A small synthetic network called Synthetic-small is generated for comparison with optimal solution. Synthetic-small contains a background graph with 20 nodes, while each community has 4 nodes and the time domain consists of 60 timestamps.

For the other two cases (Synthetic1 and Synthetic2), the network is generated in a time domain $\mathcal{T}$ of length $|\mathcal{T}| = 1000$ and the edges of each planted community in G' are generated in intervals consisting of 100 timestamps.

Synthetic1 tests the robustness of the algorithms against background noise, by varying the average degree of the background graph from 0.5 to 4, and by fixing the density of the planted graphs to 4. Synthetic2 tests the robustness of the algorithms against density of the planted communities (containing 8 nodes), by varying their density from 2 to 7, where the average degree of the background network is fixed to 2.

Comparison with Optimal Solution

We evaluate quality of solutions and running times of LSTDS versus OPTIMAL, where LSTDS applies Charikar's algorithm (denote by LSTDS-Charikar's) and Goldberg's algorithm (denoted by LSTDS-Goldberg's) as the dense subgraph method. Following [25], OPTIMAL is the dynamic programming algorithm that computes an optimal solution of k-Densest-Episodes, and it applies the Goldberg's algorithm to compute a densest subgraph of an active graph. Due to the time complexity of OPTIMAL, we consider only the Synthetic-small dataset for this comparison analysis.

The number k of intervals varies from 2 to 14. Table 1 illustrates the value of the total density and the running time of the considered methods. The results on Synthetic-small show that LSTDS is able to find near-optimal solutions, while it is significantly faster than OPTIMAL. Notice that the time required by OPTIMAL highly increases as k increases, and that thus OPTIMAL is not scalable for large graphs, even for moderate value of k.

LSTDS-Charikar's vs LSTDS-Goldberg's

We evaluate quality of the solutions and running times of LSTDS varying the dense subgraph method, that is Charikar's and Goldberg's algorithm. The goal of this comparison is to understand the impact of the dense subgraph method on the efficiency and the quality (density of the returned solutions) of LSTDS.

As reported in Table 1, LSTDS-Charikar's returns solutions that are close to those computed by LSTDS-Goldberg's. Notice that LSTDS-Charikar's is at least 25 times (for $k = 14$) and at most 35 times (for $k = 2$) faster than LSTDS-Goldberg's algorithm on this small dataset.

For this purpose, we use also the two synthetic datasets Synthetic1 and Synthetic2, averaged over 100 independent runs for $k = 3$. In Table 2, we report average, minimum and maximum density improvement[1] between these two variants of LSTDS. Using Goldberg's algorithm, the average improvement is equal to 0.3%, the maximum improvement is equal to 3% and the minimum improvement is equal to −2%. This negative result, that is the fact that by using Charikar's algorithm we obtain better solutions than using Goldberg's algorithm, is due to the fact that the candidate episodes chosen by the two variants of LSTDS can be different. Hence,

[1] The improvement is given by the difference between the densities of the solutions returned by LSTDS-Goldberg's and LSTDS-Charikar's, divided by the density of LSTDS-Charikar's.

Table 2 Comparison between LSTDS-Charikar's and LSTDS-Goldberg's algorithm

	Time (sec.)		Density
	Charikar	Goldberg	Charikar vs Goldberg
Average	0.29	9.09	0.003
Min	0.10	2.35	− 0.02
Max	0.43	16.86	0.03
solutions with equal density	–	–	64 out of 100

The values of the times are average, minimum and maximum over 100 independent synthetic networks for $k = 3$. The value of the density are averaged, minimum and maximum of improvement of LSTDS-Goldberg's with respect to LSTDS-Charikar's

in some cases LSTDS-Charikar's algorithm may choose and improve a candidate episode that is not considered by LSTDS-Goldberg's algorithm, for example because is not that of minimum density. Notice that in 64% of the examples the two variants return solutions with the same density.

As for the running time, we report in Table 2, average, minimum and maximum running time over 100 examples. As reported in Table 2, LSTDS-Charikar's algorithm with respect to LSTDS-Goldberg's algorithm is faster on average around 31 times, minimum 23.5 times and maximum 39 times.

The results of Table 2 confirm that LSTDS-Charikar's finds solutions having density close to those reported by LSTDS-Goldberg's, with significant lower running time. Due to the performance of the two variants of LSTDS, in the rest of the experimental analysis we consider only LSTDS-Charikar's algorithm. From now on, we refer to LSTDS-Charikar's algorithm simply as LSTDS.

Comparison Analysis: LSTDS vs KGAPPROX

We evaluate the performance of LSTDS and KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$, $\epsilon_1 = \epsilon_2 = 0.1$ and $\epsilon_1 = \epsilon_2 = 2$. Tables 3 and 4 report the quality of the solutions and running time of LSTDS and KGAPPROX over 100 independent runs. For this medium size synthetic datasets, we use moderate values for the k ($k = 3$ and $k = 5$).

The density and running time reported in Table 3 for the considered algorithms are averaged over 100 examples. Table 4 reports, for 100 examples, the average, minimum and maximum density improvement[2] of LSTDS with respect to the KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$, $\epsilon_1 = \epsilon_2 = 0.1$ and $\epsilon_1 = \epsilon_2 = 2$.

Table 3 shows that the solutions computed by LSTDS are denser on average than those returned by KGAPPROX, for all the values of ϵ_1 and ϵ_2 considered. Moreover, LSTDS is significantly faster than the KGAPPROX algorithm when $\epsilon_1 = \epsilon_2 = 0.01$ and $\epsilon_1 = \epsilon_2 = 0.1$. More precisely, for $k = 3$, LSTDS on average is 59.5 times faster than KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$ and 3.7 times faster than KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.1$. For $k = 5$, LSTDS is 82.5 times faster than KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$ and 4.9 times faster than KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.1$. When $\epsilon_1 = \epsilon_2 = 2$, KGAPPROX is faster than LSTDS, although the speed of the two methods are comparable. More precisely, for $k = 3$ and $k = 5$ KGAPPROX is respectively 1.7 and 1.9 times faster than LSTDS. However, notice that in this case the

Table 3 Comparison between LSTDS and KGAPPROX algorithm with $\epsilon_1 = \epsilon_2 = 0.01$, $\epsilon_1 = \epsilon_2 = 0.1$ and $\epsilon_1 = \epsilon_2 = 2$

k	LSTDS-Charikar		KGAP. $\epsilon_1 = \epsilon_2 = 0.01$		KGAP. $\epsilon_1 = \epsilon_2 = 0.1$		KGAP. $\epsilon_1 = \epsilon_2 = 2$	
	Time	Den.	Time	Den.	Time	Den.	Time	Den.
3	0.29	14.45	17.27	12.53	1.08	11.71	0.17	7.35
5	0.72	24.03	59.42	20.39	3.56	18.34	0.38	9.68

The results are average over 100 independent runs. The running time is in seconds

Table 4 Comparison of the solutions (densities) between LSTDS and KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$, $\epsilon_1 = \epsilon_2 = 0.1$ and $\epsilon_1 = \epsilon_2 = 2$

	k	KGAP. $\epsilon_1 = \epsilon_2 = 0.01$			KGAP. $\epsilon_1 = \epsilon_2 = 0.1$			KGAP. $\epsilon_1 = \epsilon_2 = 2$		
		Average	Min	Max	Average	Min	Max	Average	Min	Max
LSTDS	3	0.22	0.01	0.79	0.29	0.02	0.79	0.93	0.50	1.33
LSTDS	5	0.26	0.03	0.86	0.38	0.03	0.86	1.43	0.56	2.11

The value of the density are averaged, minimum and maximum of improvement of LSTDS with respect to the KGAPPROX over 100 independent runs. The running time is in seconds

[2] Similarly to the previous case, improvement is given by the difference between the densities of LSTDS and KGAPPROX divided by the density of KGAPPROX.

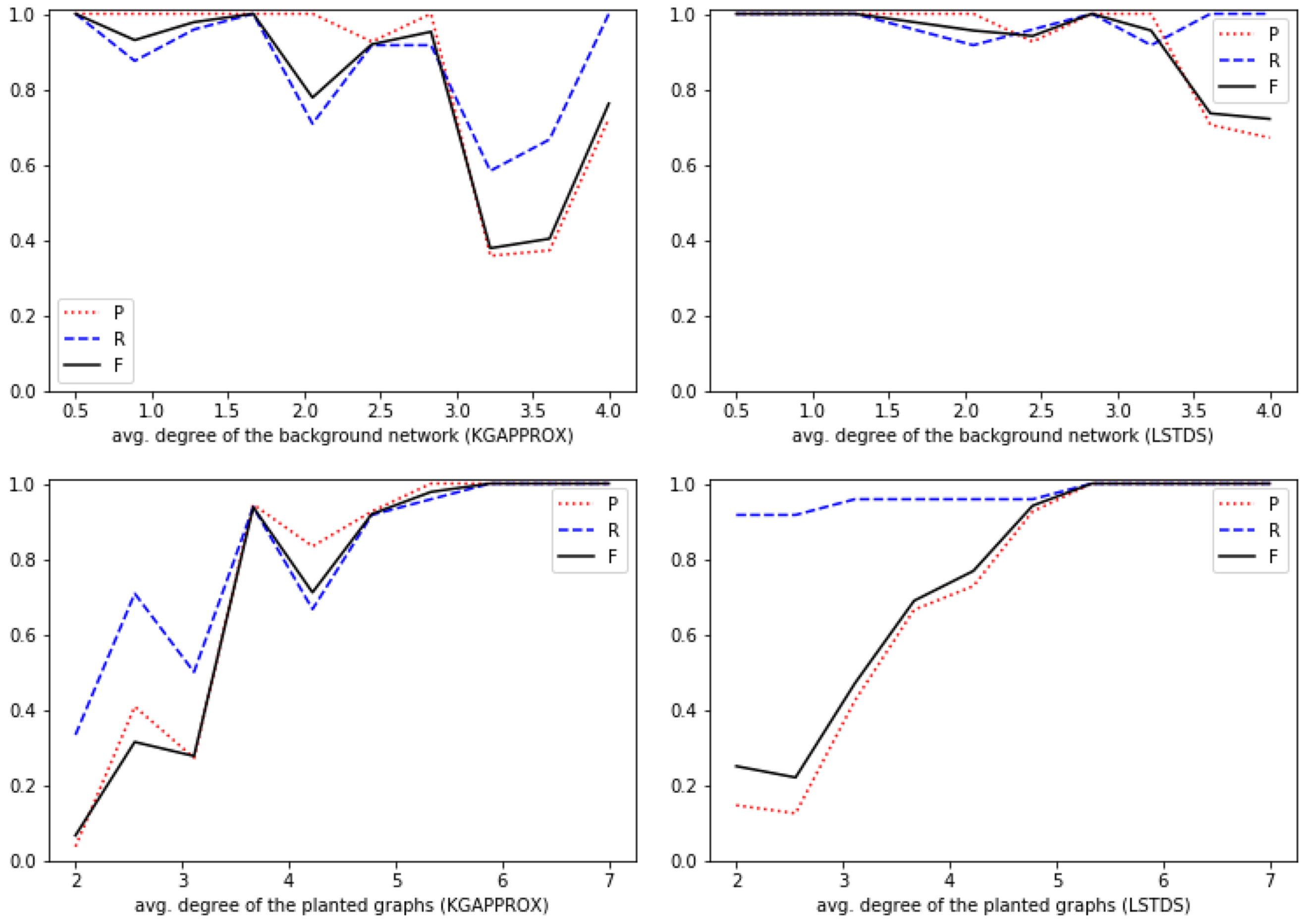

Fig. 2 Precision, recall and F-measure on synthetic datasets for $k = 3$. The two figures on the left show the performance of KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$, while the two figures on the right show the performance of LSTDS

solutions returned by LSTDS are at least 1.96 denser than those computed by KGAPPROX.

Table 4 shows that on average LSTDS improves the density of 22%, for $k = 3$, and of 26%, for $k = 5$, with respect to the best solution returned by KGAPPROX (that is when $\epsilon_1 = \epsilon_2 = 0.01$). Moreover, in the worst case the improvement of LSTDS with respect to KGAPPROX is of 1%, for $k = 3$, and of 3%, for $k = 5$.

Detecting Communities Respect to Ground-Truth

The goal of this subsection is to evaluate both algorithms by considering how well they detect planted communities. We compare LSTDS and KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$ since it is the variant of KGAPPROX that returns the best density on synthetic datasets (Synthetic1 and Synthetic2).

For both algorithms, mean precision, recall, and F-measure with respect to the ground-truth subgraphs are reported in Fig. 2, for $k = 3$, and in Fig. 3, for $k = 5$.

Figures 2 and 3 (first row) show the results for the comparison on Synthetic1, for $k = 3$ and $k = 5$, respectively. Precision, recall and F-measure of LSTDS are higher than those of KGAPPROX (in particular for $k = 5$) for all values of background noise. When the average degree of the background network increases more than 3, precision, recall and F-measure of KGAPPROX decreases significantly for $k = 3$ and $k = 5$. For LSTD, recall does not decrease significantly and precision and F-measure decrease more moderately than for KGAPPROX.

The second rows in Figs. 2 and 3, show the results for the comparison on Synthetic2, for $k = 3$ and $k = 5$, respectively. For both values of k ($k = 3$ and $k = 5$), recall of LSTDS is always higher than that of KGAPPROX for all values of average degree in planted communities. Furthermore, as reported in Figs. 2 and 3, in most of the cases precision and F-measure of LSTDS are higher than for KGAPPROX.

Real-World Datasets

We consider four different real-world datasets taken from social network platforms and emails, that have been used in [25] for testing KGAPPROX.

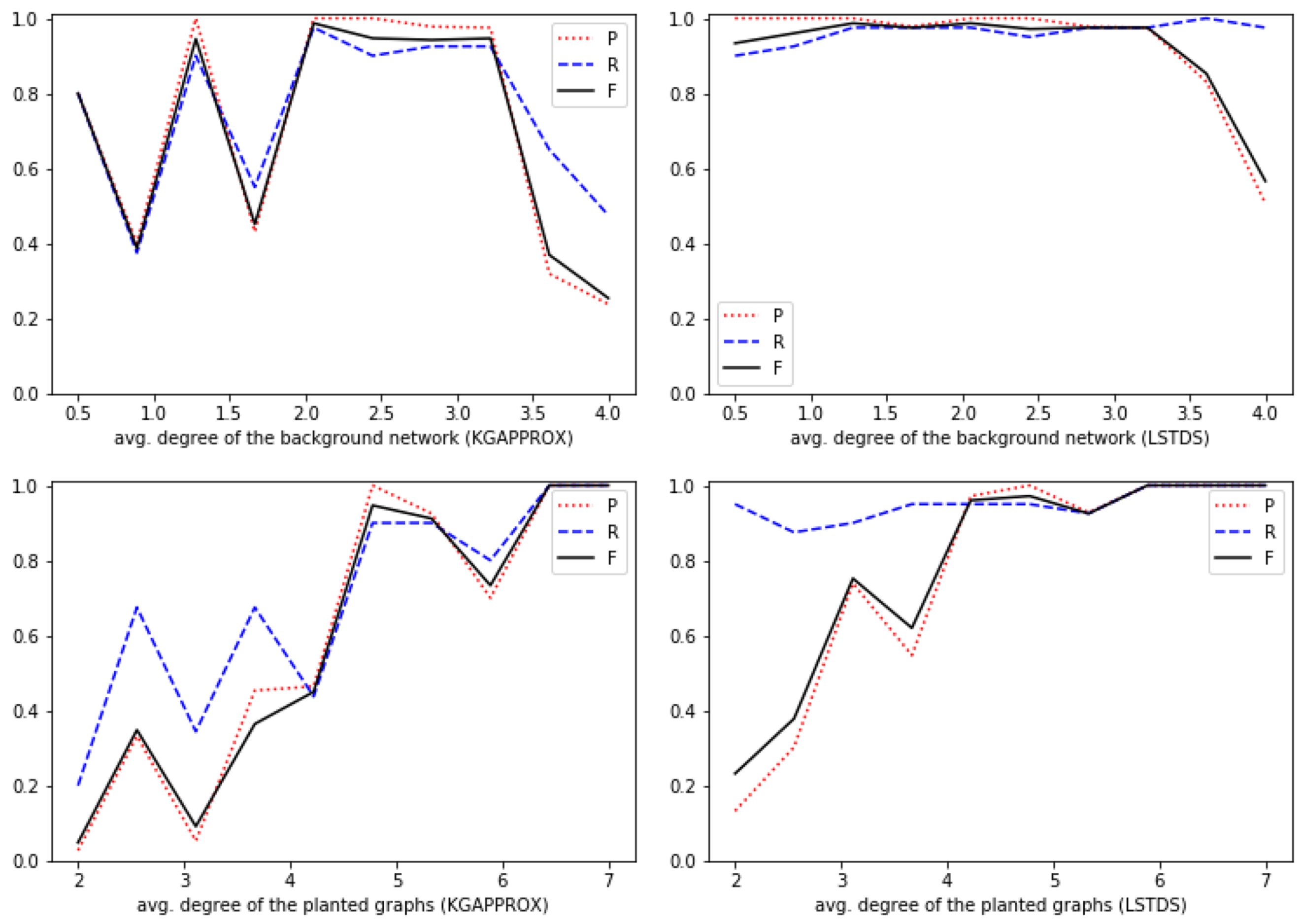

Fig. 3 Precision, recall and F-measure on synthetic datasets for $k = 5$. The two figures on the left show the performance of KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$, while the two figures on the right show the performance of LSTDS

The first dataset, Students[3], is taken from logs activity of students at the University of California, Irvine from 2004.06.27 to 2004.10.26. Nodes represent students and edges are messages exchanged between students (direction is ignored). The Students dataset contains 10000 interactions, 889 nodes and time domain $\mathcal{T}$ of length $|\mathcal{T}| = 1000$.

Enron[4] dataset was originally made public and posted by the Federal Energy Regulatory Commission[5] from 1980.01.01 to 2002.02.13. Nodes represent mostly senior managements and edges are e-mail communications between them. The Enron dataset contains 6245 interactions, 1143 nodes and time domain $\mathcal{T}$ of length $|\mathcal{T}| = 815$.

Facebook dataset is a subset of Facebook activity in the New Orleans regional community from 2006.05.09 to 2006.08.20. Nodes represent users and edges represent users posting on the walls of other users. The Facebook dataset contains 10000 interactions, 4117 nodes and time domain $\mathcal{T}$ of length $|\mathcal{T}| = 9984$.

Twitter dataset represent interaction between the users of the Twitter platform in Helsinki from 2010.07.31 to 2010.10.31. Nodes represent users and edges represent tweets that mention other users. The Twitter dataset contains 11868 interactions, 4605 nodes and time domain $\mathcal{T}$ of length $|\mathcal{T}| = 9968$.

In Table 5 we report the total density and running time of LSTDS and KGAPPROX (for the three values of ϵ_1 and ϵ_2 we have considered in this analysis) on real-world datasets. We run KGAPPROX fixing an upper bound of 10000 seconds for running time, therefore in Table 5 one entry is missing because KGAPPROX was not able to solve the problem within the considered upper bound. As shown in Table 5, only in one case KGAPPROX (when $\epsilon_1 = \epsilon_2 = 0.01$) returns a solution denser than that reported by LSTDS (Enron dataset, for $k = 5$). However, notice that, for this case, the density of the solutions returned by the two methods are very close (the solution returned by KGAPPROX is only 0.057% denser than that returned by LSTDS) and that KGAPPROX has a

[3] http://toreopsahl.com/datasets/#onlinesocialnetwork.

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

[5] https://www.ferc.gov.

Table 5 Comparison between LSTDS and KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$, $\epsilon_1 = \epsilon_2 = 0.1$ and $\epsilon_1 = \epsilon_2 = 2$ on real-world datasets for $k = 5$ and $k = 10$

k	Set	LSTDS		KGAP. $\epsilon_1 = \epsilon_2 = 0.01$		KGAP. $\epsilon_1 = \epsilon_2 = 0.1$		KGAP. $\epsilon_1 = \epsilon_2 = 2$	
		Time	Den.	Time	Den.	Time	Den.	Time	Den.
5	Students	4.66	26.37	1660.40	25.13	39.19	21.83	4.19	15.34
	Enron	3.57	41.59	2314.33	41.83	47.95	39.71	4.93	18.72
	Facebook	15.38	14.20	88.05	10.43	35.59	10.43	9.68	9.20
	Twitter	14.85	23.19	8357.91	20.44	180.88	19.96	6.63	14.45
10	Students	9.95	39.60	7696.06	38.40	163.82	34.95	11.22	20.30
	Enron	7.97	64.16	8597.59	63.27	186.19	55.93	17.63	24.55
	Facebook	32.04	25.39	223.98	15.45	100.35	15.45	32.42	14.2
	Twitter	34.17	34.36	–	–	670.83	30.07	19.78	19.45

Running time is in seconds

significantly higher running time (2314.33 vs 3.57 seconds). Moreover, LSTDS improves the solution for $k = 5$ (for these four real-world datasets) of at most 36% and on average of around 13% with respect to the best solution returned by KGAPPROX. For $k = 10$, LSTDS improves the quality of the solution at least 1.4%, at most 64% and on average around 23% with respect to the best solution returned by KGAPPROX.

As for the running time LSTDS is always faster than KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$ and $\epsilon_1 = \epsilon_2 = 0.1$, a result which confirms the comparison on synthetic dataset. For $\epsilon_1 = \epsilon_2 = 2$, KGAPPROX and LSDTS have similar running time. However, notice that in this case the solutions returned by LSTDS are denser at least 54% and at most 122%, for $k = 5$, and at least 77% and at most 161%, for $k = 10$ than those computed by KGAPPROX. The fastest variant of KGAPPROX (with $\epsilon_1 = \epsilon_2 = 2$) has lower running time than LSTDS when $k = 5$ in $\frac{3}{4}$ of the cases, when $k = 10$ in $\frac{1}{4}$ of the cases. Moreover, as reported in Table 6, for $k = 20$ in all the cases LSTDS is faster than fastest KGAPPROX with $\epsilon_1 = \epsilon_2 = 2$.

In order to further compare the speed of LSTDS and KGAPPROX, in particular for the dependency on k, we present experimental results for $k = 20$. We consider only the values $\epsilon_1 = \epsilon_2 = 2$, since, as illustrated in Table 5, when $\epsilon_1 = \epsilon_2 = 0.01$ and $\epsilon_1 = \epsilon_2 = 0.1$, KGAPPROX is always slower than LSTDS. Hence, in Table 6, we report the total density and the running time of LSTDS and KGAPPROX with $\epsilon_1 = \epsilon_2 = 2$. As for the previous cases, the solutions returned by LSTDS are much denser than those computed by KGAPPROX, on average 104%, at least 76% and at most 109% than those returned by the fastest KGAPPROX (with $\epsilon_1 = \epsilon_2 = 2$).

When $k = 20$, LSTDS is always faster than KGAPPROX. More precisely, according to Table 6, LSTDS is faster than KGAPPROX on average 2.85 times, at most 6.94 times (for the Students dataset) and at least 1.04 times (for the Twitter dataset). This result shows that the value of k has a greater impact on the running time of the KGAPPROX algorithm with respect to the running time of LSTDS.

Conclusion

In this paper, we have proposed a efficient heuristic (called LSTDS) for k-Densest-Episodes, a problem recently introduced to identify dense subgraphs in a temporal network. We have presented experimental results on synthetic and real-world datasets. First, we have compared LSTDS with OPTIMAL (the optimal algorithm for k-Densest-Episodes) on a small synthetic dataset, and the results have shown that LSTDS finds near-optimal solutions, with a significantly lower running time. We have also considered how the performances of LSTDS are influenced by the dense subgraph method applied (Goldberg's algorihtm and Charikar's algorithm). The experimental results on synthetic datasets have shown that, as expected, LSTDS with Charikar's algorithm is more efficient. Furthermore, the solutions returned by LSTDS with Charikar's algorithm are close to those returned by LSTDS with Goldberg's algorithm. Then we have compared LSTDS (with Charikar's algorithm) with KGAPPROX [25], an approximation algorithm designed for k-Densest-Episodes, on synthetic and real-world datasets. The experimental results have shown that LSTDS performs

Table 6 Comparison between LSTDS and KGAPPROX with $\epsilon_1 = \epsilon_2 = 2$ on real-world datasets for $k = 20$

k	Set	LSTDS		KGAP. $\epsilon_1 = \epsilon_2 = 2$	
		Time	Den.	Time	Den.
20	Students	23.33	63.41	36.18	30.29
	Enron	9.50	93.62	65.95	38.49
	Facebook	65.36	42.50	123.77	24.20
	Twitter	64.64	56.17	67.49	29.79

Running time is in seconds

better than KGAPPROX in terms of density and running time, thus LSTDS is a promising method to analyze large temporal networks.

There are interesting open problems related to k-Densest-Episodes and LSTDS. First, from a theoretical point of view it would be interesting to understand if LSTDS has a constant approximation factor. Moreover, it would be interesting to provide conditional lower bound on k-Densest-Episodes, as done for other graph problems [3]. From an experimental point of view, it would be interesting to consider different approaches to the preliminary phase of LSTDS and to study the performances of other local search strategies.

Author Contributions All the authors contribute to the framework definition. All the authors designed and implemented the (LSTDS) for k-Densest-Episodes problem. All the authors performed the experimental analysis. All the authors contributed to the manuscript writing. All the authors read and approved the manuscript.

Availability of data and materials Datasets are availble at: https://github.com/polinapolina/segmentation-meets-densest-subgraph.

Funding Open access funding provided by Università degli studi di Bergamo within the CRUI-CARE Agreement.

Declarations

Conflict of interest/Competing interests The authors declare that they have no competing interests.

Code availability Code is available on request.

Consent to participate We give our consent to participate.

Consent for publication We give our consent for the publication.

References

1. Angel A, Koudas N, Sarkas N, Srivastava D, Svendsen M, Tirthapura S. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. VLDB J. 2014;23(2):175–99.
2. Asahiro Y, Iwama K, Tamaki H, Tokuyama T. Greedily finding a dense subgraph. In: Algorithm Theory - SWAT '96, 5th Scandinavian Workshop on Algorithm Theory, Reykjavík, Iceland, July 3-5, 1996, Proceedings; 1996. p. 136–48. https://doi.org/10.1007/3-540-61422-2_127.
3. Backurs A, Roditty L, Segal G, Williams VV, Wein N. Towards tight approximation bounds for graph diameter and eccentricities. In: Diakonikolas I, Kempe D, Henzinger M, editors. Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25–29, 2018. ACM; 2018. p. 267–80.
4. Balalau OD, Bonchi F, Chan TH, Gullo F, Sozio M. Finding subgraphs with maximum total density and limited overlap. In: Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM. 2015. p. 379–88. https://doi.org/10.1145/2684822.2685298.
5. Charikar M. Greedy approximation algorithms for finding dense components in a graph. In: Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Proceedings. 2000. p. 84–95.
6. Coscia M, Giannotti F, Pedreschi D. A classification for community discovery methods in complex networks. Stat Anal Data Min ASA Data Sci J. 2011;4(5):512–46.
7. Dondi R, Hosseinzadeh MM, Mauri G, Zoppis I. Top-k overlapping densest subgraphs: approximation algorithms and computational complexity. J Comb Optim. 2021;41(1):80–104. https://doi.org/10.1007/s10878-020-00664-3.
8. Duhan N, Sharma A, Bhatia KK. Page ranking algorithms: a survey. In: 2009 IEEE International Advance Computing Conference. IEEE; 2009. p. 1530–1537.
9. Epasto A, Lattanzi S, Sozio M. Efficient densest subgraph computation in evolving graphs. In: Proceedings of the 24th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee. 2015. p. 300–310.
10. Ferraz Costa A, Yamaguchi Y, Juci Machado Traina A, Traina Jr C, Faloutsos C. Rsc: Mining and modeling temporal activity in social media. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM; 2015. p. 269–278.
11. Fortunato S. Community detection in graphs. Phys Rep. 2010;486(3–5):75–174.
12. Galbrun E, Gionis A, Tatti N. Top-k overlapping densest subgraphs. Data Min Knowl Discov. 2016;30(5):1134–65. https://doi.org/10.1007/s10618-016-0464-z.
13. Getoor L, Diehl CP. Link mining: a survey. ACM SIGKDD Explor Newsl. 2005;7(2):3–12.
14. Goldberg AV. Finding a maximum density subgraph. Tech. rep., Berkeley, CA, USA. 1984.
15. Holme P. Modern temporal network theory: a colloquium. Eur Phys J B. 2015;88(9):234.
16. Hosseinzadeh MM. A new heuristic to find overlapping dense subgraphs in biological networks. Proceeding of Current Trends in Theory and Practice of Computer Science p to appear. 2020.
17. Kempe D, Kleinberg J, Kumar A. Connectivity and inference problems for temporal networks. J Comput Syst Sci. 2002;64(4):820–42.
18. Kondor D, Pósfai M, Csabai I, Vattay G. Do the rich get richer? an empirical analysis of the bitcoin transaction network. PLoS One. 2014;9(2):e86197.
19. Kovanen L, Karsai M, Kaski K, Kertész J. Saramäki J (2011) Temporal motifs in time-dependent networks. J Stat Mech Theory Exp. 2011;11:P11005.
20. Li MX, Palchykov V, Jiang ZQ, Kaski K, Kertész J, Miccichè S, Tumminello M, Zhou WX, Mantegna RN. Statistically validated mobile communication networks: the evolution of motifs in european and chinese data. New J Phys. 2014;16(8):083038.
21. McGregor A, Tench D, Vorotnikova S, Vu HT. Densest subgraph in dynamic graph streams. In: Italiano GF, Pighizzini G,

Sannella D, editors. Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II, Springer, Lecture Notes in Computer Science, vol. 9235. 2015. p. 472–82. https://doi.org/10.1007/978-3-662-48054-0_39.

22. Nasir MAU, Gionis A, Morales GDF, Girdzijauskas S. Fully dynamic algorithm for top-k densest subgraphs. In: Lim E, Winslett M, Sanderson M, Fu AW, Sun J, Culpepper JS, Lo E, Ho JC, Donato D, Agrawal R, Zheng Y, Castillo C, Sun A, Tseng VS, Li C, editors. Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017. ACM; 2017. p. 1817–26. https://doi.org/10.1145/3132847.3132966.
23. Rossetti G, Cazabet R. Community discovery in dynamic networks: a survey. ACM Comput Surv (CSUR). 2018;51(2):35.
24. Rozenshtein P, Gionis A. Mining temporal networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM; 2019. p. 3225–3226.
25. Rozenshtein P, Bonchi F, Gionis A, Sozio M, Tatti N. Finding events in temporal networks: Segmentation meets densest subgraph discovery. Knowledge and Information Systems. 2019.
26. Sanli C, Lambiotte R. Temporal pattern of online communication spike trains in spreading a scientific rumor: how often, who interacts with whom? Front Phys. 2015;3:79.
27. Wackersreuther B, Wackersreuther P, Oswald A, Böhm C, Borgwardt KM. Frequent subgraph discovery in dynamic networks. In: Proceedings of the Eighth Workshop on Mining and Learning with Graphs. ACM; 2010. p. 155–162.
28. Zhao B, Wang W, Xue G, Yuan N, Tian Q. An empirical analysis on temporal pattern of credit card trade. In: International Conference in Swarm Intelligence. Springer; 2015. p. 63–70.