Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 176 (2020) 215–224

Procedia Computer Science

www.elsevier.com/locate/procedia

24th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Genetic Algorithms for Finding Episodes in Temporal Networks

Mauro Castelli[a], Riccardo Dondi[b], Mohammad Mehdi Hosseinzadeh[b,*]

[a]*NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312, Lisboa, Portugal*
[b]*Università degli Studi di Bergamo, via S. Tomaso 40, 24128, Bergamo, Italy*

Abstract

The evolution of networks is a fundamental topic in network analysis and mining. One of the approaches that has been recently considered in this field is the analysis of temporal networks, where relations between elements can change over time. A relevant problem in the analysis of temporal networks is the identification of cohesive or dense subgraphs since they are related to communities. In this contribution, we present a method based on genetic algorithms and on a greedy heuristic to identify dense subgraphs in a temporal network.

We present experimental results considering both synthetic and real-networks, and we analyze the performance of the proposed method when varying the size of the population and the number of generations. The experimental results show that our heuristic generally performs better in terms of quality of the solutions than the state-of-art method for this problem. On the other hand, the state-of-art method is faster, although comparable with our method, when the size of the population and the number of generations are limited to small values.

Keywords: Genetic algorithms; Network analysis and mining; Temporal networks; Densest subgraph.

1. Introduction

Analyzing the dynamic of real-world networks is fundamental to understand the behaviour and evolution of complex systems. Temporal networks have been introduced for this aim, since they represent how the relations between elements change over the discrete-time domain. The application of temporal networks spans different domains, notable examples being social networks, economic networks and biological networks [14, 18]. In social networks, for example, the dynamic of temporal networks has been successfully applied to identify the evolution of communities [18].

In this contribution, we study the problem of finding cohesive subgraphs inside a temporal network, a problem that has been recently proposed for the identification of real-time stories in online social networks [2, 18]. The identifica-

* Corresponding author. Tel.: +39-0352052774
E-mail address: m.hosseinzadeh@unibg.it

1877-0509
10.1016/j.procs.2020.08.023

tion of cohesive subgraphs has been a highly investigated problem in network mining (or graph mining) [10, 16, 9, 20] and a notable approach is based on finding a subgraph having largest density, that is the densest subgraph problem. The dense subgraph approach has been widely applied to analyse networks (some examples are [1, 20, 21, 22]) and has been recently extended for the identification of a set of densest subgraphs [4, 12, 8, 15, 7].

Due to its relevance in network mining, the algorithmic properties of the densest subgraph problem have been widely studied. Goldberg's algorithm [13] allows us to solve the problem in polynomial time via a reduction to the minimum cut problem. Due to its application in large-scale network mining, and due to the time complexity of Goldberg's algorithm, fast heuristics have been proposed for this problem, a notable example being a linear-time greedy algorithm with a $\frac{1}{2}$-approximate factor [3, 6].

The densest subgraph approach has been recently applied to discover dense subgraphs in temporal networks [18, 19], in the context of episode identification in online social activity. The problem introduced in [19], called k-Densest-Episodes, consider a temporal graph and looks for $k \geq 1$ densest subgraphs that belongs to disjoint time intervals. A solution of k-Densest-Episodes requires the computation of a sequence of disjoint time intervals of the given time domain and, for each one of such intervals, a densest subgraph.

Rozenshtein et al [18, 19] gives a polynomial-time algorithm for k-Densest-Episodes problem. This algorithm combines dynamic programming to identify intervals of the time domain and Goldberg's algorithm to compute a densest subgraph of a static graph in a certain interval. Unfortunately, this polynomial-time algorithm is not a viable option for large networks [18, 19], due to the complexity of both dynamic programming and Goldberg's algorithm. Thus, fast alternative approaches (heuristics) are needed for k-Densest-Episodes. Approximation algorithms have been considered in [18, 19]. More precisely, Rozenshtein et al in [18, 19] proposed an approximation algorithm, called KGAPPROX, that combines approximate dynamic programming (ApproxDP) to compute a set of disjoint intervals and an approximation algorithm for the incremental densest subgraph problem (ApprDens). ApproxDP speeds up the dynamic programming approach by considering a small set of candidates of the timestamps. The quality of approximation of ApproxDP and its running time depends on a parameter ϵ_1. ApprDens allows the computation of dense subgraph in evolving graphs. The quality of approximation of ApprDens and its running time depends on a parameter ϵ_2. The overall approximation ratio of KGAPPROX is $2(1+\epsilon_1)(1+\epsilon_2)$ [18], with $O(\frac{k^2}{\epsilon_1\epsilon_2^2}|T|mlog^2n)$ running time, for k intervals, where n and m are respectively the numbers of nodes and edges of the input graph, and T is a discrete-time domain. Notice hence that by decreasing the values of ϵ_1 and ϵ_2, the quality of the solution and the time complexity of the algorithm are both increased.

In this contribution, we consider a different approach to solve k-Densest-Episodes. Our approach, called TDGA (Temporal Densest Genetic Algorithm), combines genetic algorithms and combinatorial algorithms. A set of disjoint intervals of the time domain is computed via a genetic algorithm and, for each interval of the segmentation, the Charikar's greedy algorithm is applied to compute a dense subgraph. We show that genetic algorithms (see [17] for an introduction to genetic algorithms), appropriately bounding the size of the population and the number of generations, can be exploited to design a method whose running time is comparable with that of KGAPPROX, but improving significantly the quality (that is the density) of the returned solutions. We present variants of TDGA by varying the size of the population (values 10 and 100) and the number of generations (values 5 and 10).

We present an experimental comparison between TDGA, the optimal dynamic-programming algorithm (OPTIMAL), and KGAPPROX. The comparison between TDGA and OPTIMAL, performed on small synthetic data sets due to the complexity of OPTIMAL, shows that TDGA is significantly faster than OPTIMAL and find near-optimal solutions. The comparison between TDGA and KGAPPROX, performed on synthetic data sets and real-world networks, shows that TDGA is slower than the faster variants of KGAPPROX (although the running times of the fastest variant of TDGA and KGAPPROX are comparable). On the other hand, TDGA improves the quality of the returned solutions, in particular when compared with the fastest variant of KGAPPROX. Furthermore, the experimental results show that the running time of TDGA is less sensitive to the increase of k than the running time of KGAPPROX. The experimental results allow us to discuss how the size of the population and the number of generations influence the performance of TDGA.

The paper is organized as follows. In Section 2, we give some definitions and we introduce the k-Densest-Episodes problem. In Section 3, we present the genetic algorithms, we describe the genetic operations we consider and we briefly outline the Charikar's greedy algorithm. In Section 4, we present the experimental results on TDGA and the

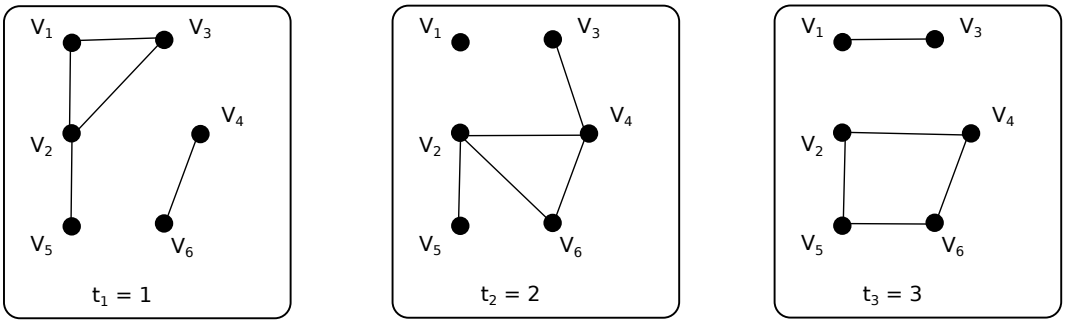

Fig. 1. An example of a temporal graph over three timestamps $t_1 = 1, t_2 = 2, t_3 = 3$, over six vertices. An episode in interval $[t_1, t_2]$ is the subgraph induced by $\{v_1, v_2, v_3, v_4, v_6\}$ of density $\frac{7}{5}$.

comparisons between TDGA, OPTIMAL and KGAPPROX. We conclude the paper with Section 5 pointing out future directions in the study on k-Densest-Episodes.

2. Definitions

First, notice that all the graphs we consider in this paper are undirect.

We introduce now the main concepts related to temporal graphs. A temporal graph is defined over a discrete time domain $T = [t_1, ..., t_q]$, where each t_i, $i \in \{1, \ldots, q\}$, is an integer, called *timestamp*, and $t_i < t_{i+1}$, for each $1 \leq i \leq q-1$.

An *interval* in T is defined as $I = [t_i, t_j]$, where $t_i, t_j \in T$, $1 \leq i, j \leq q$ and $t_i < t_j$. A set $S = \{I_1, \ldots, I_z\}$ of disjoint intervals is such that, for each intervals $I_i = [t_{i_1}, t_{i_2}]$, $I_j = [t_{j_1}, t_{j_2}]$ in S, with $1 \leq i, j \leq z$, $t_{i_2} < t_{j_1}$ or $t_{j_2} < t_{i_1}$; S is ordered if $t_{i_2} < t_{j_1}$ when $i < j$.

In the definition of temporal graph, we assume that the set of nodes does not change from timestamp to timestamp, while the set of edges may change. Since we are interested in dense graphs that belong to some interval, nodes that do not have an incident edge in that interval will not be part of any dense subgraph.

Now, we are able to define temporal graphs (see an example in Fig. 1).

Definition 2.1. *We define $G = (V, T, E)$ a temporal graph, where:*

- *V is the set of nodes*
- *$E \subseteq V \times V \times T$ is the set of temporal edges.*

A temporal edge is hence a triple (u, v, t_i), where u and v are two nodes and $t_i \in T$, $1 \leq i \leq q$, is a timestamp. It follows that if $(u, v, t_i) \in E$, then there is an edge connecting u and v at timestamp t_i. In this case we say that edge (u, v) is *active* at timestamp t_i. Consider a temporal graph $G = (V, T, E)$, an interval I in T induces an *active graph* $G_I = (V, E_I)$, where E_I is the set of active edges in a timestamp of interval I.

An *episode* is a subgraph $G'_I = (V'_I, E'_I)$ of an active graph $G_I = (V, E_I)$, that is $V'_I \subseteq V_I$ and $E'_I \subseteq E_I \cap (V'_I \times V'_I)$. The density of episode $G'_I = (V'_I, E'_I)$ is defined as:

$$dens(G'_I) = \frac{|E'_I|}{|V'_I|}.$$

Consider the active graph G_I on the interval $I = [t_1 = 1, t_2 = 2]$ in Fig. 1. An episode $G'_I = (V'_I, E'_I)$ of maximum density in I consists of $V'_I = \{v_1, v_2, v_3, v_4, v_6\}$ and $E'_I = \{(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_2, v_4), (v_2, v_6), (v_3, v_4), (v_4, v_6)\}$. The density of G_I is then $\frac{7}{5}$.

A set of episodes is called disjoint if they are defined over a set of disjoint intervals.

Now, we are able to define the problem we are interested in, called k-Densest-Episodes, introduced in [19] with the goal of finding significant episodes in a temporal graph.

Problem 1. k-Densest-Episodes
Input: *A temporal graph $G = (V, T, E)$, an integer $k \leq q$.*
Output: *A set $P = \{G'_{I_1}, \ldots, G'_{I_k}\}$ of k disjoint episodes in G such that $\sum_{j=1}^{k} dens(G'_{I_j})$ is maximized.*

We assume that a solution P of k-Densest-Episodes covers the time domain T, that is each timestamp t_i, $1 \leq i \leq q$, belongs to an episode G'_I of P.

3. Genetic Algorithms for k-Densest-Episodes

k-Densest-Episodes admits a polynomial-time of time complexity $O(kq^2|V||E|\log|V|)$ or $O(kq^2|V||E|\log\frac{|V|^2}{|E|})$ [19] (recall that q is the number of timestamps of time domain T), but its computational complexity makes it not practicable for large networks and large time domains.

The method we propose, called TDGA (Temporal Densest Genetic Algorithms), combines two different computational approaches, a genetic algorithm and a combinatorial algorithm, in order to provide denser solutions than those computed by the approximation algorithm (KGAPPROX) designed in [19] for k-Densest-Episodes. The two computational approaches are:

1. A genetic algorithm is applied to compute a set S of disjoint intervals in the time domain T.
2. Charikar's greedy algorithm is applied to compute an episode (that is a dense subgraph) for each active graph on an interval of S.

We start by describing our genetic algorithm, then we present a brief description of Charikar's greedy algorithm.

3.1. Genetic Algorithm

Consider a temporal graph $G = (V, T, E)$, we represent a set of disjoint episodes as a chromosome consisting of k integers in increasing order between t_1 and t_q, where each integer (called *point*) represents the leftmost timestamp of an interval. Hence consider, for example, chromosome $C = [c_1, \ldots, c_k]$, where c_i, $1 \leq i \leq k$, is a point. First, notice that $c_j < c_{j+1}$, for each j with $1 \leq j \leq k-1$. The i-th interval, with $1 \leq i \leq k-1$, represented by C is $[c_i, c_{i+1} - 1]$, the k-th interval is represented by $[c_k, t_q]$. Moreover, notice that $c_1 = t_1$, since each timestamp must belong to some interval.

Each chromosome C represents a set of disjoint intervals and it is associated with a vector f_C consisting of k values. The i-th value of f_C, $1 \leq i \leq k$, is the density of a dense graph, that is the density of an episode, in the i-th interval of C. The values of f_C are computed via Charikar's greedy algorithm (see the description in Section 3.2).

The genetic algorithm considers a population (a set of chromosomes) of size h. In a preliminary phase, we define an initial population of h chromosomes, by:

- Define a chromosome that represents a set of disjoint intervals that partition T such that each interval contains $\lceil \frac{1}{q} | \cup_{i=1}^{q} E_{t_i} | \rceil$ edges;
- Partition the time domain into k intervals that have the same length;
- A set of $h - 2$ chromosomes, each one computed by partitioning the time domain into k intervals defined randomly.

During the evolution, chromosomes are evolved by means of three operations: mutation, crossover, and selection. Chromosomes are evaluated via fitness function which is simply the sum of values (densities) of f_C, for a chromosome C, as the goal of k-Densest-Episodes is the maximization of the density of k episodes.

Next, we describe the evolution operators we apply.

- Mutation. We randomly mutate each point (except the first one, since $x_1 = t_1$) with probability $\frac{1}{k-1}$. Consider the case that the r-th point x_r, $2 \leq r \leq k$, is mutating, where x_r is a value in $[x_2, x_k]$. In case x_r is mutated, the new value of x_r is defined by picking a random value in the interval $[x_{r-1} + 1, x_{r+1} - 1]$, where x_{r-1}, x_{r+1} are the starting point of the $r - 1$-th and $r + 1$-th interval, respectively.
- Crossover. Crossover considers two chromosomes C_1, C_2 (parents) and computes two chromosomes C'_1, C'_2 (children). A value p is chosen randomly (with equal probability) between 2 and k and C'_1, C'_2 are defined as follows:
  - C'_1 contains the first p values of C_1 and the last $k - p$ value of C_2
  - C'_2 contains the first p value of C_2 and the last $k - p$ value of C_1.

  Notice that the value of C'_i, with $1 \leq i \leq 2$, are reordered to have points in increasing order. If some value is repeated in C'_i, only one is retained and the removed ones are replaced by picking random values in $[1, t_q]$ that do not belong to C'_i.
- Elitist selection. To ensure that the density of the computed solutions does not decrease from one generation to the subsequent [5], the h best-performing chromosomes are copied in the next population.

The procedure is iterated for g generations.

3.2. Charikar's Greedy Algorithm

Here, we briefly outline the Charikar's greedy algorithm. Given a graph (in this case an active graph $G_I = (V_I, E_I)$), Charikar's greedy algorithm approximates the densest subgraph problem as follows. Charikar's Greedy algorithm computes a set of subgraphs of G_I by iteratively removing a node of G_I. Each iteration j, with $1 \leq j \leq |V_I|$, considers in input a subgraph G_I^j of G_I and it computes the subgraph G_I^{j+1} of G_I obtained by removing a node of minimum degree from G_I^j. Notice that $G_I^1 = G_I$ and that $G_I^{|V_I|}$ consists of a single node.

Charikar's Greedy Algorithm considers then the set $\{G_I^1, ..., G_I^{|V_I|}\}$ of $|V_I|$ subgraphs computed so far. Finally, Charikar's Greedy Algorithm returns as a solution a subgraph of $\{G_I^1, ..., G_I^{|V_I|}\}$ having largest density. The algorithm has been shown to have $\frac{1}{2}$ approximation factor [6]. It is applied in several contexts (for example in [15, 12]) since it has linear-time complexity.

4. Experimental Analysis

In this section, we present the experimental results for our method, TDGA, on synthetic and real-world datasets, and we compare it with OPTIMAL and KGAPPROX. We implemented TDGA in Python (notice that also OPTIMAL and KGAPPROX were implemented in Python).

We run TDGA with different combinations of population size (h) and number of generations (g). More precisely, we consider: g=5 & h=10, g=5 & h=100, g=10 & h=10, g=10 & h=100. We limit the number of generations to 10 to allow for the analysis of large networks. Moreover, increasing the number of generations, as we have tested on some datasets, does not increase significantly the quality of the solutions.

As we will see in the next subsections, the values of parameters h and g greatly influence the running time of our method and to a less extent the quality of the returned solutions: small (large, respectively) values of h and g makes the algorithm faster (slower, respectively) and usually decreases (increases, respectively) the quality of the returned solutions.

We compare TDGA 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 these two parameters for KGAPPROX have been chosen according to [18].

The experiments were run on MacBook-Pro (OS version 10.15.3) with processor 2.9 GHzIntel Core i5 and 8GB 2133 MHz LPDDR3 of RAM, Intel Iris Graphics 550 1536 MB. Considering the stochasticity of the proposed method, for both synthetic and real networks we executed 100 independent runs. In the continuation of the paper, we reported the average (running time and performance) over the 100 runs.

Table 1. Comparison between TDGA and OPTIMAL on *Synthetic-small*. The results are averaged over 100 independent runs. The running time is in seconds.

		$k = 2$	$k = 4$	$k = 6$	$k = 8$	$k = 10$	$k = 12$	$k = 14$
OPTIMAL	Time	9.23	80.64	137.39	247.77	526.66	664.67	911.79
	Den.	5.28	9.71	12.89	14.53	17.14	19.62	22.52
TDGA								
g=5&h=10	Time	0.20	0.63	0.77	0.91	1.07	1.20	1.31
	Den.	5.28	9.40	12.33	13.87	15.82	18.06	21.17
g=5&h=100	Time	1.84	6.18	7.46	9.01	10.62	11.62	13.07
	Den.	5.28	9.59	12.76	14.30	16.49	18.64	21.56
g=10&h=10	Time	0.35	1.21	1.41	1.66	1.93	2.17	2.42
	Den.	5.28	9.41	12.36	13.83	15.68	18.23	21.23
g=10&h=100	Time	3.34	11.99	13.75	16.28	20.00	21.62	24.64
	Den.	5.28	9.60	12.89	14.52	16.77	19.14	22.00

4.1. Synthetic datasets

Following Rozenshtein et al [18], we generate temporal graphs consisting of a background network, based on Erdős-Rényi model, with k planted communities, that is k planted subgraphs. Erdős-Rényi model [11] is a well-known model for generating random graphs. Given a set of nodes, each pair of nodes is connected with an edge with independent probability, depending on a parameter. The background network G includes all nodes over the discrete-time domain T. We vary edges distribution according to the average degree of the background graph G (below we specify how density changes in different experiments). The background graph contains k dense subgraphs (k planted communities) induce a subgraph of G. These k communities are defined in non-overlapping intervals and they have the same density.

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

First, a small synthetic network called *Synthetic-small* is generated for comparison with OPTIMAL, the optimal algorithm for k-Densest-Episodes. Indeed, due to the computational complexity of OPTIMAL, we have to limit the comparison to a small dataset. *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 datasets (*Synthetic1* and *Synthetic2*), networks are generated in a time domain T of length 1000 and the edges of each planted community 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 the 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.

4.1.1. TDGA vs OPTIMAL

We evaluate the running times and the density of returned solutions of TDGA with different parameter values versus OPTIMAL. Recall that TDGA applies Charikar's greedy approximation algorithm to compute a dense subgraph of an active graph. OPTIMAL is based on dynamic programming and applies the Goldberg's algorithm to compute a densest subgraph of an active graph [18].

The number k of intervals varies from 2 to 14. Table 1 shows the total density and the running time of TDGA and OPTIMAL. The results show that TDGA can find near-optimal solutions, while it is significantly faster than OPTIMAL. For small value of k ($k = 2, 4, 6, 8$), TDGA returns solutions of density very close (in some cases equal) to the density of solutions returned by OPTIMAL. When k increases, the solutions returned by TDGA are less dense compared with OPTIMAL (the maximum decreasing in density with respect to OPTIMAL is of 7.95% when $k = 12$, g=5&h=10).

Table 2. Comparison between TDGA and KGAPPROX. The results are averaged over 100 independent runs. The running time is in seconds.

		$k = 3$	$k = 5$
TDGA			
g=5&h=10	Time	3.22	5.08
	Den.	14.58	24.26
g=5&h=100	Time	30.94	47.84
	Den.	14.59	24.35
g=10&h=10	Time	6.74	9.86
	Den.	14.58	24.26
g=10&h=100	Time	54.86	86.66
	Den.	14.59	24.37
KGAPPROX			
$\epsilon_1 = \epsilon_2 = 0.01$	Time	17.27	59.42
	Den.	12.53	20.39
$\epsilon_1 = \epsilon_2 = 0.1$	Time	1.08	3.56
	Den.	11.71	18.34
$\epsilon_1 = \epsilon_2 = 2$	Time	0.17	0.38
	Den.	7.35	9.68

The running time required by OPTIMAL highly increases as k increases from 9.23 second ($k = 2$) to 911.79 second (when $k = 14$). This confirms that OPTIMAL is not scalable for large graphs, as it is not applicable even for medium size graphs and moderate value of k, as reported in [19]. The performance of TDGA is less influenced by the value of k. In the worst case (g=10&h=100). In the worst case (g=10&h=100), the running time of TDGA for $k = 14$ increases approximately 738% with respect to the case $k = 2$, while the running time of OPTIMAL increases of around 9879%.

In this small dataset, we start also to analyze the performance of TGDA when varying parameters g and h. As expected the densest solutions are always returned with parameters g=10 & h=100. The densities returned by TDGA with g=5 & h=100 are close to the case g=10 & h=100. This confirms that increasing the size of the population allows the computation of denser solutions, even when reducing the number of generations. The running time is heavily affected by the size of the population: the case g=10 & h=100 is at least 16 times slower than g=5 & h=10, while the decreasing of the density of this latter variant is at most 5.66% with respect to the case g=10 & h=100.

4.1.2. TDGA vs KGAPPROX

We evaluate TDGA with the same parameters considered in the previous subsection and we compare its performances with those of KGAPPROX, with values $\epsilon_1 = \epsilon_2 = 0.01$, $\epsilon_1 = \epsilon_2 = 0.1$ and $\epsilon_1 = \epsilon_2 = 2$. For this comparison, we use the two synthetic datasets *Synthetic1* and *Synthetic2*, over 100 independent runs. For this medium size synthetic datasets, we use moderate values of k ($k = 3$ and $k = 5$).

Table 2 reports the density of the returned solutions and the running time of TDGA and KGAPPROX averaged over 100 independent runs. These experimental results show that the solutions returned by TDGA in all the cases considered are denser on average than those computed by KGAPPROX. The fastest variant of KGAPPROX ($\epsilon_1 = \epsilon_2 = 2$) returns solutions that have roughly half density for $k = 3$ and 40% of the density for $k = 5$ with respect to the less effective variant of TDGA (g=5 & h=10). The variant of KGAPPROX returning densest solutions ($\epsilon_1 = \epsilon_2 = 0.01$) shows a decreasing of at least 14% ($k = 3$) and 15.9% ($k = 5$) in density with respect to the less effective variant of TDGA (g=5 & h=10). Notice also that the quality of the solutions returned by TDGA, when applied to this datasets, is not changing significantly when varying the values of g and h. Indeed, for $k = 3$, the average values of the returned solutions are basically identical. When $k = 5$ the decreasing of the density of the less effective case is limited to 0.45%.

Considering the running time, TDGA with g=5 & h=10 and g=10 & h=10 is significantly faster than the KGAPPROX algorithm with $\epsilon_1 = \epsilon_2 = 0.01$ and comparable with $\epsilon_1 = \epsilon_2 = 0.1$ (especially for $k = 5$), but slower than KGAPPROX with $\epsilon_1 = \epsilon_2 = 2$ (in this case the density of the solutions returned by TDGA is significantly higher than the solutions computed by TDGA). More precisely, for $k = 3$, TDGA with g=5 & h=10 on average is 5.4 times faster than KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$, 3 times slower than KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.1$ and 19 times slower than

KGAPPROX with $\epsilon_1 = \epsilon_2 = 2$. In this case the solutions returned by TDGA with g=5 & h=10 are 1.7 times denser than the best solutions of KGAPPROX ($\epsilon_1 = \epsilon_2 = 0.01$). For $k = 5$, TDGA with g=5 & h=10 on average is faster than KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$ (11.7 times) and slower than the other variants of KGAPPROX (0.7 times slower for $\epsilon_1 = \epsilon_2 = 0.1$, 9.4 times slower for $\epsilon_1 = \epsilon_2 = 2$). Notice that in these cases (in particular for$\epsilon_1 = \epsilon_2 = 2$) the solutions returned by KGAPPROX are considerably less dense.

The results on this synthetic datasets confirm also that the value of k has a greater impact on the running time of KGAPPROX, in particular for small values of ϵ_1 and ϵ_2, with respect to the running time of TDGA. For $\epsilon_1 = \epsilon_2 = 0.01$, the average running time for $k = 5$ is 3.44 times of the running time for $k = 3$. For TDGA, the increase in the running time of the case $k = 5$ compared to $k = 3$ is at most 1.58 times

4.2. Real-world datasets

We analyze the performance of TDGA considering four different real-world datasets taken from social network platforms and emails, considered in [18]. The first dataset considered, called *Students*[1], contains logs activity of students of the University of California, Irvine, between 2004.06.27 and 2004.10.26. In this network, nodes represent students and edges represent messages exchanged between students. This network contains 10000 interactions, 889 nodes, and time domain T consists of 1000 timestamp. The second data, called *Enron*[2], contains messages (made public by the Federal Energy Regulatory Commission[3]) exchanged from 1980.01.01 to 2002.02.13. In this network, nodes represent managements, while edges represent e-mail communications between managements. This network contains 6245 interactions, 1143 nodes, and time domain T consists of 815 timestamp. The third dataset, *Facebook*, consists of a portion of Facebook activity in the New Orleans regional community from 2006.05.09 to 2006.08.20. In this network, nodes represent users, while edges user posting on the walls of other users. The Facebook network contains 10000 interactions, 4117 nodes, and time domain T consists of 9984 time stamps. The fourth dataset, *Twitter*, represent interactions between Helsinky users of Twitter between 2010.07.31 and 2010.10.31. In this network, nodes represent users and edges represent tweets that mention other users. The Twitter dataset contains 11868 interactions, 4605 nodes and time domain T consists of 9968 time stamp.

In Table 3, we present the running time and the densities of the solutions returned by TDGA (for the four combinations of parameters) and KGAPPROX (for the three values of ϵ_1 and ϵ_2 considered) on real-world datasets. We run the algorithms fixing an upper bound of 10000 seconds for running time. Table 3 has a blank entry for KGAPPROX (*Twitter* for $k = 10$) since it was not able to return the result within the considered upper bound.

As shown in Table 3, the density returned by different variants of TDGA has small variations. The decreasing in density of the less dense returned solution with respect to the densest returned solution is on average of 4.3%, with a maximum of 9.19% (for the Facebook datasets with $k = 5$). Except in one case (*Students* for $k = 5$), the results of TDGA when g=10&h=100 are always denser than the other combinations of parameters of TDGA. In this latter case, notice that the highest value (when g=5&h=100) is only marginally. However, while increasing the size of populations leads to a small improvement in the quality (density) of the solution, the running time is heavily affected. The slowest variant (g=10&h=100) is on average 17.84 slower than the fastest variants (g=5&h=10), at least 13.61 slower and at most 21.64.

The comparison between TDGA and KGAPPROX on these real-world datasets shows that TDGA always produces significantly denser solutions than KGAPPROX. Even the less dense solutions returned by TDGA are denser than the densest solution returned by KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$. For the other variants of KGAPPROX the gap is more significant. Notice that for the case $\epsilon_1 = \epsilon_2 = 2$, the gap is particularly relevant, with a density decrease of at least 55%, at most 122% and on average around 77% for $k = 5$ and at least 72%, at most 155% and on average around 100% for $k = 10$.

As for the running time, TDGA (even in the slowest case when g=10&h=100) is always faster than KGAPPROX with $\epsilon_1 = \epsilon_2 = 0.01$, the variant of KGAPPROX that returns densest solutions. On the other hand, the fastest variant

[1] http://toreopsahl.com/datasets/#onlinesocialnetwork
[2] http://www.cs.cmu.edu/~./enron/
[3] https://www.ferc.gov

Table 3. Comparison between TDGA and KGAPPROX on real-world datasets for $k = 5$ and $k = 10$. The results are averaged over 100 independent runs. The running time is in seconds.

		Students		**Enron**		**Facebook**		**Twitter**	
		$k = 5$	$k = 10$	$k = 5$	$k = 10$	$k = 5$	$k = 10$	$k = 5$	$k = 10$
TDGA									
g=5&h=10	Time	37.85	42.36	24.48	31.82	102.84	131.20	106.80	129.53
	Den.	26.05	39.48	41.66	62.60	14.23	24.38	23.30	34.61
g=5&h=100	Time	288.54	347.49	246.44	273.29	1099.87	1234.25	1082.30	1237.83
	Den.	26.71	40.43	42.65	63.41	15.65	25.46	23.90	35.49
g=10&h=10	Time	49.08	60.57	42.28	47.64	181.37	201.95	183.24	199.42
	Den.	26.56	38.67	41.66	62.43	14.26	24.36	23.34	34.62
g=10&h=100	Time	515.30	612.60	454.02	511.83	2225.32	2450.81	2255.06	2404.92
	Den.	26.68	40.50	42.97	65.24	15.67	25.86	24.09	35.99
KGAPPROX									
$\epsilon_1 = \epsilon_2 = 0.01$	Time	1660.40	7696.06	2314.33	8597.59	88.05	223.98	8357.91	–
	Den.	25.13	38.40	41.83	63.27	10.43	15.45	20.44	–
$\epsilon_1 = \epsilon_2 = 0.1$	Time	39.19	163.82	47.95	186.19	35.59	100.35	180.88	670.83
	Den.	21.83	34.95	39.71	55.93	10.43	15.45	19.96	30.07
$\epsilon_1 = \epsilon_2 = 2$	Time	4.19	11.22	4.93	17.63	9.68	32.42	6.63	19.78
	Den.	15.34	20.30	18.72	24.55	9.20	14.2	14.45	19.45

Table 4. Comparison between TDGA with g=5&h=10 and KGAPPROX with $\epsilon_1 = \epsilon_2 = 2$ on real-world datasets for $k = 20$. The results are averaged over 100 independent runs. Running time is in seconds.

		Students	**Enron**	**Facebook**	**Twitter**
		$k = 20$	$k = 20$	$k = 20$	$k = 20$
TDGA					
g=5&h=10	Time	40.97	38.01	145.91	153.20
	Den.	62.66	94.39	42.06	55.96
KGAPPROX					
$\epsilon_1 = \epsilon_2 = 2$	Time	36.18	65.95	123.77	67.49
	Den.	30.29	38.49	24.20	29.79

of KGAPPROX (with $\epsilon_1 = \epsilon_2 = 2$) is always faster than TDGA for all values of h and g, but in this case, as reported before, the quality of the solutions of KGAPPROX is considerably lower compared with TDGA.

For a larger value of k ($k = 20$), we consider only the fastest variants of the two methods, TDGA with g=5&h=10 and KGAPPROX with $\epsilon_1 = \epsilon_2 = 2$. The results in Table 4 show that the running time of the two methods are comparable, and only in one case TDGA is significantly slower than KGAPPROX (2.27 times slower for the Twitter dataset). Notice also that in one case (Enron dataset) TDGA is faster than KGAPPROX. Moreover, notice that the gap in the running time is reduced with respect to the case $k = 5$ and $k = 10$, where TDGA was on average 10.28 and 4.05, respectively, slower than KGAPPROX (with the considered parameters' values). This confirms that the running time of KGAPPROX is more influenced by the value of k than TDGA. The comparison of the density shows that also for $k = 20$ TDGA returns denser solutions. More precisely, TDGA returns solutions that are at least 1.75 denser than KGAPPROX (Facebook dataset) and at most 2.42 denser that KGAPPROX (Enron dataset).

5. Conclusion

In this paper, we considered a computational approach, called TDGA, based on genetic algorithm and Charikar's greedy algorithm for the k-Densest-Episodes problem in temporal networks. We presented experimental results on synthetic and real-world data sets, where we varied two parameters of the genetic algorithm, the size of the populations

and the number of generations. The experimental results showed that by increasing the size of the population, the quality of the solution increases, but this comes at the cost of a significant amount of computational time.

We also presented an experimental comparison with a previous computational approach (KGAPPROX) for k-Densest-Episodes, showing that our approach produces denser solutions and, even if it runs slower than the fastest variant of KGAPPROX, it has a comparable running time when k increases.

Future work on TDGA include expanding the experimental work, considering more combinations of parameters g and h and larger datasets of synthetic networks, varying for example the background graph model. Moreover, we plan to extend the experimentation to larger real networks. Finally, another future direction is the application of other evolutionary computational approaches to k-Densest-Episodes and the comparison with TDGA.

References

[1] Andersen, R., Chellapilla, K., 2009. Finding dense subgraphs with size bounds, in: Avrachenkov, K., Donato, D., Litvak, N. (Eds.), Algorithms and Models for the Web-Graph, 6th International Workshop, WAW 2009, Barcelona, Spain, February 12-13, 2009. Proceedings, Springer. pp. 25–37. doi:10.1007/978-3-540-95995-3_3.

[2] Angel, A., Koudas, N., Sarkas, N., Srivastava, D., Svendsen, M., Tirthapura, S., 2014. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. The VLDB Journal 23, 175–199.

[3] Asahiro, Y., Iwama, K., Tamaki, H., Tokuyama, T., 1996. Greedily finding a dense subgraph, in: Algorithm Theory - SWAT '96, 5th Scandinavian Workshop on Algorithm Theory, Reykjavík, Iceland, July 3-5, 1996, Proceedings, pp. 136–148. doi:10.1007/3-540-61422-2_127.

[4] Balalau, O.D., Bonchi, F., Chan, T.H., Gullo, F., Sozio, M., 2015. 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, pp. 379–388. doi:10.1145/2684822.2685298.

[5] Baluja, S., Caruana, R., 1995. Removing the genetics from the standard genetic algorithm, in: Prieditis, A., Russell, S.J. (Eds.), Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995, Morgan Kaufmann. pp. 38–46.

[6] Charikar, M., 2000. Greedy approximation algorithms for finding dense components in a graph, in: Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Proceedings, pp. 84–95.

[7] Dondi, R., Hermelin, D., 2020. Computing the k densest subgraphs of a graph. CoRR abs/2002.07695. URL: https://arxiv.org/abs/2002.07695, arXiv:2002.07695.

[8] Dondi, R., Hosseinzadeh, M.M., Mauri, G., Zoppis, I., 2019a. Top-k overlapping densest subgraphs: Approximation and complexity. Proceeding of ICTCS 2504.

[9] Dondi, R., Mauri, G., Sikora, F., Zoppis, I., 2019b. Covering a graph with clubs. J. Graph Algorithms Appl. 23, 271–292. doi:10.7155/jgaa.00491.

[10] Epasto, A., Lattanzi, S., Sozio, M., 2015. 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. pp. 300–310.

[11] ErdÅs, P.; Rnyi, A., 1959. On random graphs. i. Publicationes Mathematicae 6, 290 – 297.

[12] Galbrun, E., Gionis, A., Tatti, N., 2016. Top-k overlapping densest subgraphs. Data Min. Knowl. Discov. 30, 1134–1165. doi:10.1007/s10618-016-0464-z.

[13] Goldberg, A.V., 1984. Finding a Maximum Density Subgraph. Technical Report. University of California at Berkeley. Berkeley, CA, USA.

[14] Holme, P., 2015. Modern temporal network theory: a colloquium. The European Physical Journal B 88, 234.

[15] Hosseinzadeh, M.M., 2020. Dense subgraphs in biological networks, in: International Conference on Current Trends in Theory and Practice of Informatics, Springer. pp. 711–719.

[16] Kempe, D., Kleinberg, J., Kumar, A., 2002. Connectivity and inference problems for temporal networks. Journal of Computer and System Sciences 64, 820–842.

[17] Mitchell, M., 1996. An introduction to genetic algorithms. Complex adaptive systems, MIT press, Cambridge (Mass.).

[18] Rozenshtein, P., Bonchi, F., Gionis, A., Sozio, M., Tatti, N., 2019. Finding events in temporal networks: Segmentation meets densest subgraph discovery. Knowledge and Information Systems .

[19] Rozenshtein, P., Gionis, A., 2019. Mining temporal networks, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM. pp. 3225–3226.

[20] Sozio, M., Gionis, A., 2010. The community-search problem and how to plan a successful cocktail party, in: Rao, B., Krishnapuram, B., Tomkins, A., Yang, Q. (Eds.), Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010, ACM. pp. 939–948. doi:10.1145/1835804.1835923.

[21] Tatti, N., Gionis, A., 2015. Density-friendly graph decomposition, in: Gangemi, A., Leonardi, S., Panconesi, A. (Eds.), Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015, ACM. pp. 1089–1099. doi:10.1145/2736277.2741119.

[22] Zou, Z., 2013. Polynomial-time algorithm for finding densest subgraphs in uncertain graphs, in: Proceedings of Internation Workshop on Mining and Learning with Graphs, pp. 1–7.