Finding dynamic dense subgraphs

POLINA ROZENSHTEIN, Helsinki Institute for Information Technology, Aalto University, Finland
NIKOLAJ TATTI, Helsinki Institute for Information Technology, Aalto University, Finland
ARISTIDES GIONIS, Helsinki Institute for Information Technology, Aalto University, Finland

Online social networks are often defined by considering interactions of entities at an *aggregate* level. For example, a *call graph* is formed among individuals who have called each other at least once; or at least k times. Similarly, in social-media platforms we consider *implicit social networks* among users who have interacted in some way, e.g., have made a conversation, have commented to the content of each other, etc. Such definitions have been used widely in the literature and they have offered significant insights regarding the structure of social networks. However, it is obvious that they suffer from a severe limitation: they neglect the precise time that interactions among the network entities occur.

In this paper we consider *interaction networks*, where the data description contains not only information about the underlying topology of the social network, but also the exact time instances that network entities interact. In an interaction network an edge is associated with a time stamp, and multiple edges may occur for the same pair of entities. Consequently, interaction networks offer a more fine-grained representation, which can be leveraged to reveal otherwise hidden dynamic phenomena.

In the setting of interaction networks we study the problem of discovering *dynamic dense subgraphs* whose edges occur in *short time intervals*. We view such subgraphs as fingerprints of dynamic activity occurring within network communities. Such communities represent groups of individuals who interact with each other in specific time instances, for example, a group of employees who work on a project and whose interaction intensifies before certain project milestones. We prove that the problem we define is **NP**-hard, and we provide efficient algorithms by adapting techniques for finding dense subgraphs. We also show how to speed-up the proposed methods by exploiting *concavity* properties of our objective function, and by the means of *fractional programming*. We perform extensive evaluation of the proposed methods on synthetic and real datasets, which demonstrates the validity of our approach and shows that our algorithms can be used to obtain high-quality results.

CCS Concepts: •**Mathematics of computing** → **Graph algorithms;** •**Information systems** → **Data mining;** •**Theory of computation** → **Dynamic graph algorithms;**

General Terms: Theory, Algorithms

Additional Key Words and Phrases: Graph mining, social-network analysis, dynamic graphs, time-evolving networks, interaction networks, dense subgraphs, community discovery

ACM Reference Format:
Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis, 2016, Finding dynamic dense subgraphs. *ACM Trans. Knowl. Discov. Data.* V, N, Article A (January YYYY), 30 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Searching for communities is one of the most well-studied problems in social-network analysis. A number of different methods has been proposed, employing a diverse set of algorithmic tools, such as, agglomerative approaches, min-cut formulations, random walks, spectral methods, and more.

This work was supported by Academy of Finland grant 118653 (ALGODAN)

DOI: http://dx.doi.org/10.1145/0000000.0000000

On the other hand, it has been observed that large networks do not exhibit clear community structure [Gleich and Seshadhri 2012; Leskovec et al. 2010]. The lack of well-defined communities is typically contributed to the high degree of inter-connectivity, and the existence of overlapping communities. The phenomenon is aggravated by the fact that existing community-detection methods do not take into account the temporal dimension of the data, which may provide additional cues about the underlying community structure.

Our work is motivated by the observation that, nowadays, there are sufficiently rich datasets so as it is possible to analyze not only the underlying network topology but also the exact time of interactions among the network entities. Our hypothesis is that the temporal dimension of network interactions can be utilized to reveal information about the network structure and dynamics, which otherwise may be hidden. To be more concrete, consider the following examples.

Example 1: A group of researchers across many different institutions are collaborating on a large project. The members of the group go along with their everyday activities, often unrelated to the project. However, once every few weeks or months, before deliverable deadlines or project meetings, there is a lot of interaction among the group members.

Example 2: A group of twitter users are interested in technology products, and they are very active in blogging reviews and commenting the posts of each other. Their interaction is sparse, but it sustains over a long time, and it intensifies significantly after the release of a new product.

The main point in these two examples is that the communities of interest *are not isolated*. Their members interact with each other, but they also interact with others outside the community. If one ignores the temporal dynamics and considers only the static network topology, the communities are hidden and it is impossible to discover them. It is only when considering the exact interaction times that the communities become detectable: in both of the above examples, many interactions occur among the community members, but in a number of relatively short time intervals.

In this paper we formalize the idea exemplified above. We consider *interaction networks* for which we assume that the exact time of all interactions is known. Examples of such networks include *call graphs* in telecommunications, *email networks*, *mention networks* in social media, *collaboration networks*, and more. Thus, interaction networks are abundant in many application domains.

In the setting of interaction networks we study the problem of discovering *dynamic dense subgraphs* whose edges occur in *short time intervals*. or equivalently, are *temporally compact*. We view such subgraphs as fingerprints of dynamic activity occurring within network communities. We show that this problem is **NP**-hard. This hardness result should be contrasted with the fact that finding the densest subgraph in static graphs is a polynomially-time solvable problem.

We then propose two different algorithms for discovering dynamic dense and temporally-compact subgraphs. Our algorithms combine ideas from methods for finding dense subgraphs and for solving covering problems. We also show how to exploit the *concavity* property of our objective function, and how to apply *fractional programming*, in order to speed up our algorithms considerably. In particular, for one of the proposed algorithms we are able to achieve almost linear running time, making it the algorithm of choice for large datasets.

To evaluate the proposed methods we conduct an extensive experimental evaluation using synthetic and real-world datasets. Our experiments demonstrate the effectiveness of the proposed algorithms, as well as the validity of our hypothesis. Namely, we

show that it is possible to find subgraphs that satisfy the requirements we set: there are dense interactions that occur within a number of short time intervals.

Finding dense subgraphs in static graphs is a well-studied graph-mining problem and it has a wide range of applications, ranging from biology [Bader and Hogue 2003; Sharan and Shamir 2000], to fraud and link-spam detection [Beutel et al. 2013; Gibson et al. 2005], to social-media mining [Angel et al. 2012]. Our works extends the problem of dense-subgraph finding in temporal networks, and consequently provides support for applications of dense-subgraph finding in the temporal setting. For example, Angel et al. [2012] show how finding dense subgraphs is used for real-time story identification in twitter. Using our method in the same application domain would allow to reveal news stories that may not be detected when ignoring the temporal dimension; it also helps in providing additional information about the temporal extent of a news story. As a second example, Beutel et al. [2013] show that dense bipartite subgraphs in page–like data reveal attempts to inflate like counts in a fraudulent manner. As such artificial page likes are typically set by bots, it is likely that those page likes have a temporal footprint, making our method appropriate for detecting fraud in this setting.

The rest of the paper is organized as follows. In Section 2 we present our notation and some preliminaries, which in Section 3 we formally define our problem and establish its complexity. In Section 4 we present our algorithms, which we evaluate using synthetic and real-world datasets in Section 5. We discuss the related work in Section 6 and Section 7 is a short conclusion.

2. PRELIMINARIES AND NOTATION

An *interaction network* $G = (V, E)$ consists of a set of n nodes V and a set of m time-stamped interactions E between pairs of nodes

$$E = \{(u, v, t)\}, \quad \text{such that} \quad u, v \in V \quad \text{and} \quad t \in \mathbb{R}.$$

We consider that interactions are *undirected*. More than one interaction may take place between a pair of nodes, with different time stamps. Conversely, more than one interaction may take place at the same time, between different pairs of nodes.

For an interaction network $G = (V, E)$ we associate the set of edges $\pi(E)$ to be the pairs of nodes for which there is at least one interaction (one may think of π as "projecting" the edges of the interaction network along the time axis)

$$\pi(E) = \{(u, v) \in V \times V \mid (u, v, t) \in E \text{ for some } t\}.$$

Given an interaction network $G = (V, E)$, the network $\pi(G) = (V, \pi(E))$ is a standard graph with no time stamps on its edges. We refer to $\pi(G)$ as the *topology network* of G or as the *underlying network* of G. The cardinality of the set of edges $\pi(E)$ is denoted by r.

Given an interaction network $G = (V, E)$ and a subset of nodes $W \subseteq V$, we define the *induced interaction network* $G(W) = (W, E(W))$, such that $E(W)$ consists of the interactions whose both end-points belong in W, that is,

$$E(W) = \{(u, v, t) \in E \mid u, v \in W\}.$$

We assume that all interactions take place during a time interval $[t_m, t_M]$ of duration $\Delta = t_M - t_m$. We consider that the interval $[t_m, t_M]$ contains ℓ discrete time points with granularity δ, that is $\ell \leq \Delta/\delta$. We also define the *span* of any time interval $T = [s, e] \subseteq [t_m, t_M]$ as $\text{span}(T) = e - s$.

We define a *time-interval set* $\mathcal{T}$ to be a collection of non-overlapping time intervals, $\mathcal{T} = (T_1, \ldots, T_k)$. The span of $\mathcal{T}$ is the sum of the spans of its intervals,

$$\operatorname{span}(\mathcal{T}) = \sum_{i=1}^{k} \operatorname{span}(T_i)\,.$$

Given an interaction network $G = (V, E)$ and a time interval $T = [s, e]$ we define the *spliced interaction network* $G(T) = (V, E(T))$, where $E(T)$ are the interactions that occur in T,

$$E(T) = E([s, e]) = \{(u, v, t) \in E \mid s \leq t \leq e\}\,.$$

The above notion can be extended in a straightforward manner so as to define the spliced interaction network with respect to a set of time intervals $\mathcal{T} = (T_1, \ldots, T_k)$. This is achieved by collecting edges from individual time intervals, that is, $G(\mathcal{T}) = (V, E(\mathcal{T}))$, where $E(\mathcal{T}) = \bigcup_{i=1}^{k} E(T_i)$.

The concepts of *induced interaction network* and *spliced interaction network* provide two different ways to select subsets of interaction networks; one is based on subsets of nodes and the other is based on time intervals. The definition of dynamic dense subgraphs, which is the central concept of this paper, relies on these two strategies for selecting subsets of edges. In particular, for an interaction network $G = (V, E)$, a subset of nodes W, and a set of time intervals $\mathcal{T}$, we define a *dynamic dense subgraph* $G(W, \mathcal{T})$ as the subgraph that consists of the nodes in W and the set of interactions among the nodes in W that occur within $\mathcal{T}$. In more formal terms, $G(W, \mathcal{T})$ is defined to be the spliced interaction network $H(\mathcal{T})$, where H is the induced interaction network $G(W)$.

To measure the quality of a dynamic dense subgraph we rely on the notion of *density*. We first recall the definition of density as defined for static graphs and we also review the *densest-subgraph problem*.

Given a static graph $H = (V, F)$, i.e., the edges F do not have time stamps, the *density* $d(H)$ of H is defined by

$$d(H) = \frac{2\,|F|}{|V|}. \tag{1}$$

The density $d(H)$ can be interpreted as the *average degree* of the nodes in H.

PROBLEM 2.1 (DENSEST SUBGRAPH). *Given a static graph $H = (V, F)$, find the set of nodes $W \subseteq V$ that maximizes the density $d(H(W))$.*

Finding the densest subgraph is polynomially-time solvable [Goldberg 1984]. Furthermore, there is a linear-time factor-2 approximation algorithm [Asahiro et al. 2000; Charikar 2000]. The algorithm deletes iteratively a node with the lowest degree, obtaining a sequence of subgraphs. Among those subgraphs the algorithm returns the one with the highest density. We should point that there exist alternative definitions for density. A popular variant is the propotion of edges, $\frac{|F|}{\binom{|V|}{2}}$. Unlike with the $d(H)$, maximizing this density leads to the maximum clique problem, which is not only **NP**-hard but is inapproximable to a factor of $\mathcal{O}\big(n^{1-\epsilon}\big)$ for any $\epsilon > 0$ [Feige 2004]. Due to these computational differences we prefer the average-degree density over the proportion of edges.

3. PROBLEM FORMULATION

Our goal is to discover subgraphs that are dense and whose interactions occur in short time intervals. Given an interaction network $G = (V, E)$, we aim to find a set of nodes W and a set of time intervals $\mathcal{T}$, such that the subgraph $G(W)$ is relatively dense

within $\mathcal{T}$. To ensure that the subgraph $G(W)$ is temporally compact, we impose two types of constraints on the time-interval set $\mathcal{T}$:

(i) constraints on the number of intervals of $\mathcal{T}$; and

(ii) constraints on the total length of $\mathcal{T}$.

We discuss these two constraints shortly. For the problem of finding dense dynamic subgraphs, we also assume a *quality score* $q(W, \mathcal{T}; G)$ that measures the density of the subgraph $G(W, \mathcal{T})$ of the interaction network G.

PROBLEM 3.1. *Consider an interaction network $G = (V, E)$. Given a set of nodes $W \subseteq V$ and a set of time intervals $\mathcal{T}$ let $q(W, \mathcal{T}; G)$ be a* quality score *that measures the density of a dynamic subgraph $G(W, \mathcal{T})$. Assume also we are given a budget K on the number of time intervals, and a budget B on the total time span. Our goal is to find the set of nodes W and the set of time intervals $\mathcal{T}$ that maximize*

$$q(W, \mathcal{T}; G)\,, \text{ such that } |\mathcal{T}| \leq K \text{ and } \operatorname{span}(\mathcal{T}) \leq B.$$

The first constraint states that we can have at most K intervals while the second constraint requires that the total duration is at most B. One may ask if both of these constraints are necessary. As we will explain below both constraints are required, otherwise Problem 3.1 has contrived solutions.

Let us first discuss why a budget constraint (B) on the total time span is needed. This is a consequence of the natural assumption that the quality score $q(W, \mathcal{T}; G)$ increases with the span of $\mathcal{T}$: indeed, increasing the total span allows to take more edges and have a denser structure. Thus, without a budget constraint on the total span we can just take $\mathcal{T}$ to be the whole time interval that covers all the edges. Such a solution, however, does not capture the intuition of dynamic subgraphs that we aim to discover. Instead we aim to find subgraphs whose interactions occur concentrated in short time periods.

Next, we discuss why the constraint on the total number of time intervals (K) is also necessary. To see this, note that if we allow to select an unlimited number of time intervals, then we can select any subset of interactions $E' \subseteq E$. Indeed, any interaction $(u, v, t) \in E'$ can be selected by adding in $\mathcal{T}$ a time interval of the form $(t - \epsilon, t + \epsilon)$, with $\epsilon \to 0$. Since any subset of interactions can be selected, any subgraph can be found as a solution with no regard to the temporal aspect of the data. In other words, the constraint on the number of intervals is necessary to impose time-continuity on the solutions found.

Regarding the score function used to assess the quality of a subgraph, our proposed measure is the *density of the corresponding topology network*, after restricting to node set W and time-interval set $\mathcal{T}$. Namely, we set

$$q(W, \mathcal{T}; G) = d(\pi(G(W, \mathcal{T})))\,,$$

where the density function $d(\cdot)$ is given by Equation (1). In other words, we count twice the number of interactions that occur between nodes of W within time intervals in $\mathcal{T}$, and normalize by $|W|$.

We proceed to establish the complexity of the problem of finding a dense dynamic subgraph in interaction networks (Problem 3.1). The proof of Proposition 3.1 is provided in the Appendix.

PROPOSITION 3.1. *Problem 3.1 is **NP**-hard.*

ALGORITHM 1: Iterative algorithm for finding a dense and temporally-compact subgraph

Input: Interaction network G
Output: Subgraph $G(W, \mathcal{T})$
$\mathcal{T}_0 \leftarrow$ initial sets of time intervals;
$i \leftarrow 0$;
while *(convergence;* $i \leftarrow i + 1$*)* **do**
  $W_{i+1} \leftarrow$ solution to Problem 4.1 given $\mathcal{T}_i$;
  $\mathcal{T}_{i+1} \leftarrow$ solution to Problem 4.2 given W_{i+1};
end
return $G(W_i, \mathcal{T}_i)$;

4. ALGORITHMS FOR DISCOVERING DENSE AND TEMPORALLY-COMPACT SUBGRAPHS

To solve Problem 3.1 we propose an iterative method. Our algorithm considers the two components of the solution, the node set W and the time interval set $\mathcal{T}$, and optimizes each one in an alternating fashion, while keeping the other fixed.

Both steps of our alternating optimization method give rise to interesting computational problems. One problem reduces to finding the densest subgraph, and the other is related to coverage, and it is **NP**-hard. Next we formalize the two problems of our alternating optimization method.

PROBLEM 4.1. *Assume an interactive network $G = (V, E)$ with a quality score q. Given a set $\mathcal{T}$ of time intervals, find a set of nodes W that maximizes $q(W, \mathcal{T}; G)$.*

PROBLEM 4.2. *Assume an interactive network $G = (V, E)$, a budget K on the number of time intervals, a budget B on the total time span, and a quality score q. Assume that a set of nodes W is provided as input. Find a set $\mathcal{T}$ of time intervals that maximizes*

$$q(W, \mathcal{T}; G)\,, \text{ such that } |\mathcal{T}| \leq K \text{ and } \operatorname{span}(\mathcal{T}) \leq B.$$

The proposed algorithm starts from an initial time interval set $\mathcal{T}_0$, and obtains a solution $(W, \mathcal{T})$ by iteratively solving the two problems defined above until convergence. Pseudo-code of the method is given in Algorithm 1. As one may expect the iterative algorithm does not provide a guarantee for the quality of the solution that it returns. However, as it is stated by the following proposition, whose proof is given in the Appendix, it has the desirable property that both of the alternating optimization problems return the correct component of the solution if they obtain as input the other component correctly.

PROPOSITION 4.1. *Let $(W^*, \mathcal{T}^*)$ be an optimal solution to Problem 3.1 for an interaction network G. Then (i) W^* is an optimal solution to Problem 4.1 given G and $\mathcal{T}^*$, and (ii) $\mathcal{T}^*$ is an optimal solution to Problem 4.2 given G and W^*.*

We should stress that we cannot solve Problem 4.2 exactly, and thus, there is no automatic guarantee that the score increases during the while loop. However, we can easily ensure termination by stopping the algorithm when the quality of the solution does not increase. Notice that our objective function takes at most $\mathcal{O}(nr)$ unique values, where $r = |E(\pi(G))|$, and so the algorithm is guaranteed to stop after at most so many steps. This, however, is a pessimistic upper bound; in our experiments the algorithm terminates in a much smaller number of steps, usually less than 20.

In the next two sections, 4.1 and 4.2, we present in detail our solution for the two subproblems of Algorithm 1. In Section 4.5 we discuss the initialization of the algorithm.

4.1. Finding an optimal set of nodes

We start with Problem 4.1 where the goal is to find an optimal set of nodes W given a set $\mathcal{T}$ of time intervals. Assume that we are given a set $\mathcal{T}$ of time intervals, and let $H = \pi(G(\mathcal{T}))$ be the topology network for the interactions that occur within $\mathcal{T}$, that is, the topology network of the interaction network spliced by $\mathcal{T}$. Note that

$$q(W, \mathcal{T}; G) = d(H(W)) .$$

Consequently, finding the optimal set of nodes is equivalent to the densest-subgraph problem (Problem 2.1) on the (static) graph H. It follows that finding the optimal set of nodes W, given time interval set $\mathcal{T}$, can be done in polynomial time. In our implementation, we use the linear-time algorithm of Charikar [2000], which, as outlined in Section 2, offers a factor-2 approximation guarantee.

4.2. Finding an optimal set of time intervals

We now present our solutions for the second subproblem of the iterative algorithm, namely, finding an optimal set of time intervals for a given set of nodes. Unfortunately, even this subproblem remains **NP**-hard. The proof of this claim is a simplified version of the proof of Proposition 3.1.

We view the problem of finding optimal time intervals as an instance of a *maximum-coverage with multiple budgets* (MCMB) problem.

PROBLEM 4.3 (MCMB). *Given a ground set $U = \{u_1, \ldots, u_m\}$ with weighted elements $w(u_i)$, a collection of subsets $\mathcal{S} = \{S_1, \ldots, S_k\}$, p cost functions c_i mapping each subset of $\mathcal{S}$ to a positive number, and p budget parameters B_i, find a subset $\mathcal{P} \subseteq \mathcal{S}$ maximizing*

$$\sum_{u \in X} w(u), \quad \textit{such that } X = \bigcup_{S \in \mathcal{P}} S, \quad \textit{and } \sum_{S \in \mathcal{P}} c_i(S) \leq B_i, \textit{ for all } i = 1, \ldots, p.$$

When $p = 1$, the problem is the standard budgeted maximum coverage. The problem is **NP**-hard but there exists an approximation algorithm by Khuller et al. [1999] that achieves $(1 - 1/e)$ approximation ratio. However, this algorithm requires to enumerate all 3-subset collections, making it infeasible in practice.

The optimization problem can be also viewed as an instance of maximizing a submodular function under multiple linear constraints. Kulik et al. [2009] presented a polynomial algorithm that achieves $(1-\epsilon)(1-1/e)$ approximation ratio. Unfortunately, this algorithm is not practical even for modest ϵ.

To see how finding a set of time intervals is related to maximum coverage, consider as ground set the set of edges $\pi(E(\mathcal{T}))$, that is, interactions that occur in $\mathcal{T}$ without the time stamps, and for each time interval $T \in \mathcal{T}$ create a subset S_T containing all edges whose corresponding interactions occur in T. Since all covered edges count equally, we use a uniform weight function $w(e) = 1$, for all $e \in \pi(E(\mathcal{T}))$. There are two cost functions $c_1(T) = 1$ and $c_2(T) = \mathrm{span}(T)$. The first budget constraint enforces the number of allowed time intervals to stay below K, while the second budget enforces the time-span constraint.

Thus, we need to solve the MCMB problem, defined above, with two budget constraints. The approximation algorithms of Khuller et al. [1999] and Kulik et al. [2009] can be used for solving the MCMB problem, however, as we discussed, both of these algorithms are highly impractical.

To address this scalability issue, we propose two alternative algorithms that are very efficient and can be used in practice for finding communities in large interaction

networks. Both algorithms are inspired by the standard greedy approach for maximum coverage. The difference between the two proposed approaches is on how they try to satisfy the budget constraints. The first approach sets a parameter that controls the amount of violation of one constraint, and optimizes this parameter with binary search, while the second approach incorporates both budget constraints into a single gain/cost ratio during the greedy step.

4.3. Binary search approach

Our first approach is based on incorporation of time constraint into the cost function. Consider the following optimization problem.

PROBLEM 4.4. *Given a graph G, a subset of nodes W, a budget K on the number of time intervals, and a budget B on the total time span, find a set of intervals $\mathcal{T}$ to maximize*

$$Q_\alpha(\mathcal{T}) = q(W, \mathcal{T}; G) - \alpha \cdot \mathrm{span}(\mathcal{T}), \tag{2}$$

such that $|\mathcal{T}| \leq K$.

Note that we do not enforce any time-budget constraint. If $\alpha = 0$, the solution consists of a single time interval that spans the entire dataset. On the other hand, if α is set to be large, the best set $\mathcal{T}$ will consist of K intervals having ϵ span, with $\epsilon \to 0$, and each of these intervals encloses a single interaction. In fact, as it is shown in the following proposition, and proven in the Appendix, the time span of the optimal solution decreases as α increases.

PROPOSITION 4.2. *Consider parameters α_1 and α_2 with $\alpha_1 < \alpha_2$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the solutions of Problem 4.4 for α_1 and α_2, respectively. Then* $\mathrm{span}(\mathcal{T}_1) \geq \mathrm{span}(\mathcal{T}_2)$.

Ideally, if we can solve Problem 4.4 optimally, we can use binary search to find the smallest α such that the time span of the solution does not exceed the budget. As we do not have an exact solver for Problem 4.4, we apply a greedy approach where in each step we find a single time interval that maximizes the objective function (2). In each step of the greedy algorithm, all time intervals are candidates to be selected, but long intervals are penalized by the objective function. We then apply binary search to find α that produces a feasible solution, i.e., see Algorithm 2, named BA. The initial upper bound α_u must provide a solution that fits the budget, while the lower bound α_l must give a solution violates the budget. As initial lower bound we chose $\alpha_l = 0$, while the upper bound α_u is chosen according to the following Proposition.

PROPOSITION 4.3. *Let $\alpha_u = 2|E(\pi(G(W)))|(B|W|)^{-1} \leq 2r/B$. Let $\mathcal{T}$ be a set of intervals such that $Q_{\alpha_u}(\mathcal{T}) \geq 0$. Then the span of the set of intervals $\mathcal{T}$ satisfies the budget constraint B, i.e.,* $\mathrm{span}(\mathcal{T}) \leq B$.

The set of intervals for Problem 4.4 are found greedily. Since any interval with zero duration will have a non-negative impact, the solution given by the greedy will always be non-negative. Consequently, as initial value for α_u, we can use the value given in Proposition 4.3

In order to monitor convergence we can use the following proposition and terminate the iteration of Algorithm 2 when $\alpha_u - \alpha_l$ becomes small enough.

PROPOSITION 4.4. *Let α^* be the smallest parameter value for which the corresponding solution $\mathcal{T}^*$ satisfies* $\mathrm{span}(\mathcal{T}^*) \leq B$. *Define $\epsilon = 1/(\delta\Delta^2 w)$, where δ is the granularity of time stamps, Δ is the duration of the whole time interval that the interaction network spans, and $w = |W|$.*

ALGORITHM 2: BA: discovers a time interval set for a fixed set of nodes W.

Input: Set of nodes $W \subseteq V$, budget B, budget K, α_l, α_u
Output: Optimal feasible set $\mathcal{T}$ for Problem 4.4
Solve Problem 4.4 for α_u and obtain $\mathcal{T}(\alpha_u)$;
$\alpha_c \leftarrow (\alpha_l + \alpha_u)/2$;
while *not converged* **do**
  Solve Problem 4.4 for α_c and obtain $\mathcal{T}_c$;
  Check if $\text{span}(\mathcal{T}(\alpha_c)) > B$ and update $\alpha_l, \alpha_u, \alpha_c, \mathcal{T}(\alpha_u)$;
end
return $\mathcal{T}(\alpha_u)$;

Consider α_l and α_u with $\alpha_l < \alpha_u \leq \alpha_l + \epsilon$. Let $\mathcal{T}_l$ and $\mathcal{T}_u$ be the solutions of Problem 4.4 for α_l and α_u, respectively. If $\text{span}(\mathcal{T}_u) \leq B \leq \text{span}(\mathcal{T}_l)$, then $q(\mathcal{T}_u) = q(\mathcal{T}^*)$.

Proposition 4.4 states that the optimal set of time intervals $\mathcal{T}_u$ that corresponds to the parameter value α_u satisfies the time-budget constraint, and its quality score is no worse than any other time-interval set that satisfies the time-budget constraint. Furthermore, this optimality condition is satisfied as soon as the difference $\alpha_u - \alpha_l$ becomes smaller than ϵ. Consequently, the binary search requires $\mathcal{O}(\log(\alpha_u/\epsilon)) = \mathcal{O}(\log(nr\Delta\delta))$ steps. We should stress that since in practice we are using a greedy heuristic to solve Problem 4.4, we do not have guarantees that we have converged to the correct value of α. Nevertheless, we can still use the same stopping condition to guarantee the termination of the algorithm in a logarithmic number of steps.

For solving Problem 4.4 for a fixed α, we apply the greedy method outlined before. Namely, we build the interval set $\mathcal{T}$ by adding one interval at a time, while selecting the interval that maximizes the gain in the objective function Q_α. A naïve implementation of the method requires testing all possible intervals. This algorithm, denoted as BASIC-BA, has quadratic running time with respect to the number of time stamps.

However, it is possible to speed-up considerably this basic greedy algorithm. The speed-up relies on applying an effective pruning scheme, which is based on the fact that the gain function in the greedy step is *concave*.

More specifically, consider a function $f : U \to \mathbb{R}$, where U is the set of all intervals, $U = \{(a, b) \in \mathbb{R}^2 \mid a \leq b\}$. We say that f is *concave* if it satisfies the *concave Monge condition*

$$f(a, d) - f(b, d) \leq f(a, c) - f(b, c),$$

for $a \leq b \leq c \leq d$. This condition is referred to as concavity by some authors, such as Wilber [1988] and Galil and Park [1992], while other authors refer to it as convexity. The condition is also analogous to submodularity, but we do not use this term here, as submodularity is typically used to characterize functions over sets of elements rather than intervals. Note also that as f is a function over intervals, rather than real variables, the term concavity here does not refer to the usual definition of concave functions.

Recall that our goal is to greedily approximate Problem 4.4. More specifically, assume that we have already discovered $\mathcal{T}$ and write $\delta_\alpha(T) = Q_\alpha(\mathcal{T} \cup T)$. Our next interval is the one that optimizes $\delta_\alpha(T)$. The next proposition is proven in the Appendix.

PROPOSITION 4.5. *The gain function $\delta_\alpha(T)$ is concave.*

Given a concave function f, define $e(b) = \arg\max_j f(b, j)$, the end point of the optimal interval that starts at b. Clearly, if we know $e(b)$ for every starting point b we can find the global optimum by a single scan and a linear number of calls to the function f.

Concave functions have the following key property:

ALGORITHM 3: FIND-INT, Fast discovery of an optimal time interval using concavity-based pruning.

Input: A concave function f over intervals, ordered sequence of all possible border points $P = (p_i)$, a set of starting points $B \subseteq P$
Output: optimal end point $e(b)$ for each $b \in B$
if $|B| = 1$ **then**
  compute $e(b)$, where $b \in B$;
else
  $J \leftarrow \{b_{2i} \in B\}$;
  $e \leftarrow$ FIND-INT(f, P, J);
  foreach $b = b_{2i+1} \in B$ **do**
    $e(b) \leftarrow \arg\max_j \{f(b, j) \mid e(b_{2i}) \leq j \leq e(b_{2i+2})\}$;
  end
end
return e;

PROPOSITION 4.6. *Consider a concave function f defined over intervals. Then $e(i) \leq e(j)$, whenever $i < j$.*

The proof of this proposition is provided by Aggarwal et al. [1987], and it is a cornerstone of the SMAWK algorithm, a linear-time matrix-searching algorithm. We cannot use directly the SMAWK algorithm since it assumes that we can evaluate the function f in constant time, and this assumption does not hold in our case. However, we can still use some key ideas from the SMAWK algorithm, which will give us a significant speed-up over the naïve case.

The algorithm is recursive, and it computes $e(b)$ for each $b \in B$, where B is a set of points and it is given as input to the recursive call. If B contains only one point, the algorithms makes a full scan to find the optimal interval. Otherwise, it first computes $e(b)$, recursively, for every *even* starting point $b \in B$. Once $e(b)$ is computed for all even points $b \in B$, the algorithm proceeds by computing $e(b)$ for every *odd* starting point $b \in B$. Here the algorithm uses the fact that $e(b_{2i}) \leq e(b_{2i+1}) \leq e(b_{2i+2})$ to limit the search space and avoid unnecessary checks.

The scheme is used to find the next best interval T, given a set of intervals $\mathcal{T}$ found so far. We call the algorithm FIND-INT, and it is outlined in Algorithm 3. We apply FIND-INT with δ_α as the quality function and the set of all edge time-stamps as the candidate set of borders. We then ran this algorithm K times to find an interval set $\mathcal{T}$. The resulting algorithm to solve Problem 4.4, enhanced with the fast scheme FIND-INT to discover a single optimal time interval, is named FAST-BA.

Let us now analyze the computational complexity of FIND-INT. We claim that the complexity is $\mathcal{O}(m \log m \log n)$. Since the number of starting points is halved with every recursive call, there are at most $\mathcal{O}(\log m)$ calls of FIND-INT.

If the input set B contains only one point, then FIND-INT needs $\mathcal{O}(m \log n)$ time for a full scan; the $\log n$ factor is needed for computing the number of unique edges in a single interval. Otherwise, we need to make sure that the for-loop in the else-branch takes only $\mathcal{O}(m \log n)$ time. We can do this by maintaining a multiset of all the interactions that are in a interval that is currently tested: whenever we increase the ending point, we add an interaction, and whenever we increase the stating point, we remove interactions. Since a single interaction can be added and/or removed only once, at most, this gives $\mathcal{O}(m)$ updates with a single update needing $\mathcal{O}(\log n)$ time. Consequently, this gives us $\mathcal{O}(m \log m \log n)$ time. The total running time of FAST-BA is $\mathcal{O}(Km \log m \log n \log(\delta \Delta n r))$, or $\tilde{\mathcal{O}}(Km)$, using the notation $\tilde{\mathcal{O}}$ to supress logarithmic factors.

In practice, as we will see in our experimental evaluation, FAST-BA is several orders of magnitude faster than BASIC-BA.

4.4. Greedy approach

Consider for a moment the classic *set-cover* problem. The standard greedy approach for this problem is to select the set that covers the most elements *per unit of cost*. Motivated by this idea, we suggest the following greedy approach: given a currently selected set of time intervals $\mathcal{T}$, we find the interval T, that satisfies the constraint $\text{span}(\mathcal{T} \cup \{T\}) \leq B$, while maximizing the ratio

$$\frac{q(W, \mathcal{T} \cup \{T\}; G) - q(W, \mathcal{T}; G)}{\max(x, y)},$$
$$\text{where } x = \frac{1}{K - |\mathcal{T}|} \text{ and } y = \frac{\text{span}(T)}{B - \text{span}(\mathcal{T})}.$$

The numerator in the ratio is the number of new edges covered with the interval T. The denominator is the maximum of two quantities, x and y, representing the two constraints of our problem, on the number of intervals and on the total time span. Both x and y are normalized so that they are equal to 1 if adding T will cap the corresponding constraint. By taking the maximum of the ratios we consider the constraint that is closer to be capped and penalize the ratio accordingly. We refer to this greedy approach as GA.

A brute-force interval search is limited to the intervals that do not violate the time budget. We refer to this brute-force search as BASIC-GA.

To simplify our notation, given a set of time intervals $\mathcal{T}$, we define $B' = B - \text{span}(\mathcal{T})$ to be the remaining span budget, and $K' = K - |\mathcal{T}|$ the number of additional intervals that can be added to $\mathcal{T}$. We also rewrite the cost function as

$$Q_g(T) = \frac{g(T, \mathcal{T}, W, G)}{c(T, B', K')},$$

where g is defined as

$$g(T, \mathcal{T}, W, G) = q(W, \mathcal{T} \cup \{T\}; G) - q(W, \mathcal{T}; G),$$

and c is defined as,

$$c(T, B', K') = \begin{cases} \frac{1}{K'}, & \text{if } \text{span}(T) \leq \frac{B'}{K'} \\ \frac{\text{span}(T)}{B'}, & \text{otherwise.} \end{cases}$$

For further convenience of notation, we assume that G, W, and $\mathcal{T}$ are fixed, and we write $g(T)$ for $g(T, \mathcal{T}, W, G)$ and $c(T)$ for $c(T, B', K')$.

During each iteration of the greedy process we solve the following problem.

PROBLEM 4.5. *Consider a graph G, a set of subset of nodes W, budgets K and B, and a set $\mathcal{T}$ of already selected time intervals, such that $|\mathcal{T}| < K$ and $\text{span}(\mathcal{T}) \leq B$. The goal is to find an interval T, such that $\text{span}(T) \leq B'$ and $Q_g(T)$ is maximized.*

Unfortunately, the function Q_g is not concave and we can not use techniques as in Algorithm 3 to speed-up the search for an optimal interval. However, we can rewrite the problem as a series of concave optimization problems using the *fractional programming* technique [Dinkelbach 1967]. Next, we briefly outline this technique, and we discuss how it can be applied in our setting. We start by restating Problem 4.5 in a more abstract formulation.

ALGORITHM 4: Fractional programming approach for solving Problem 4.5.

Input: Initial β_u, set of nodes W, discovered intervals $\mathcal{T}$, and remaining budgets B' and K'
Output: Optimal next interval T_l
$\epsilon \leftarrow$ guantity defined in Proposition 4.9;
Solve Problem 4.7 for β_u and obtain T_u;
$\beta_l \leftarrow g(T_u)/c(T_u)$;
Solve Problem 4.7 for β_l and obtain T_l;
while $h(\beta_l, T_l) > 0$ **and** $g(T_l) \neq g(T_u)$ **and** $\beta_u \geq \beta_l + \epsilon$ **do**
  $\beta_c \leftarrow (\beta_l + \beta_u)/2$;
  Solve Problem 4.7 for β_c and obtain T_c;
  if $h(T_c, \beta_c) < 0$ **then** $\beta_u \leftarrow \beta_c$, $T_u \leftarrow T_c$;
  else $\beta_l \leftarrow \beta_c$, $T_l \leftarrow T_c$;
  $\beta_l \leftarrow \max(g(T_c)/c(T_c), g(T_l)/c(T_l))$;
  Solve Problem 4.7 for β_l and obtain T_l;
end
return $\arg\max(g(T_l)/c(T_l), g(T_u)/c(T_u))$;

PROBLEM 4.6. *Let $g, c : \mathbb{R}^2 \to \mathbb{R}$ be two continuous real-valued functions from intervals, and let r be a number. We want to find an interval T, such that* $\mathrm{span}(T) \leq r$ *and the ratio $g(T)/c(T)$ is maximized.*

Fractional programming provides a solution to Problem 4.6 by considering the following associated maximization problem.

PROBLEM 4.7. *Given a non-negative number $\beta \in \mathbb{R}$, let $h(T, \beta) = g(T) - \beta \cdot c(T)$. Find an interval T that maximizes $h(T, \beta)$ such that* $\mathrm{span}(T) \leq B'$*, where B' is a span budget.*

Given a number $\beta \in \mathbb{R}$ let us denote by T_β the solution to Problem 4.7. Problems 4.6 and 4.7 are intimately related through the following proposition.

PROPOSITION 4.7 ([DINKELBACH 1967]).[1] *An interval T^* is a solution to Problem 4.6 if and only if T^* is a solution to Problem 4.7 that satisfies $h(T^*, \beta^*) = 0$ with $\beta^* = \frac{g(T^*)}{c(T^*)}$.*

Proposition 4.7 allows us to solve Problem 4.6 (and thus, its special case, Problem 4.5) by searching for an interval T that solves Problem 4.7 with $h(T, \beta^*) = 0$, assuming that the value β^* is known. Searching for such an interval T can be done efficiently due to the following proposition — which we prove in the Appendix.

PROPOSITION 4.8. *The gain function $h(T, \beta)$ is concave with respect to T.*

Thus we can use pruning techniques similar to the one discussed in the previous section. However, the value β^* is not known. To find β^*, we use the fact that $h(T_\beta, \beta)$ is a monotonically decreasing function of β. This fact is proven by Dinkelbach [1967]. The monotonicity can be used to find β^* by binary search. Starting with β_l and β_u we perform binary search until we can guarantee the optimality condition. To get slightly better theoretical guarantees, we can simultaneously use the iteration suggested by Dinkelbach [1967]: we solve T_l for β_l and set the new β_l to be $g(T_l)/c(T_l)$.

The approach outlined above and shown in Algorithm 4 is named FAST-GA. The properties of FAST-GA algorithm are stated in the following proposition, whose proof is given in the Appendix.

[1] Dinkelbach [1967] assumes that q is continuous which is not the case here. However, the same proof holds for any q and any arbitrary domain.

PROPOSITION 4.9. *Let W be the currently selected nodes, let $w = |W|$ and set*

$$p = \binom{w}{2} - |\pi(E(\mathcal{T}))|$$

to be the maximum number of uncovered edges. Let B' and K' be the remaining budgets. Let δ be the granularity of time points, and set $\epsilon = 1/(wK'B'\delta)$. Then Algorithm 4 with the input of $\beta_u = p/wK'$ solves Problem 4.5 in $\mathcal{O}(\min(p, \log wK'B'\delta))$ iterations.

We should point out that the last two statements in the while-loop of Algorithm 4 are not mandatory. Removing these two lines will speed-up the algorithm in practice but the theoretical guarantee for computational complexity given in Proposition 4.9 will increase to $\mathcal{O}(\log wK'B'\delta)$. A single iteration needs $\mathcal{O}(m \log m \log n)$ time, and this yields a total running time for FAST-GA of $\mathcal{O}(m \log m \log n \min(r, \log \delta nKB))$. Ignoring the logarithmic factors, the running time is $\tilde{\mathcal{O}}(m)$, i.e., linear with respect to the number of interactions.

4.5. Initialization

The quality of the solution depends on the set of time intervals $\mathcal{T}_0$ used as initial seed. Consider an optimal solution $(W, \mathcal{T})$, with $\mathcal{T} = (T_1, \ldots, T_K)$, which achieves density d^*. It follows that there is one single time interval $T \in \mathcal{T}$, for which the optimal set of nodes W has density at least d^*/K on $\pi(G(T))$. This observation motivates us to limit ourselves to consider only time interval sets of size 1. Assuming large computational power, one could test every possible time interval as a seed, consequently run the iterative algorithm, and return the best solution found. There are $\mathcal{O}(m^2)$ such intervals, which is polynomial.

If running the algorithm $\mathcal{O}(m^2)$ times is expensive, we can select J random intervals, run the iterative algorithm for each of those random intervals, and return the best solution found. In our experiments we evaluate the effect of the number of random seeds J to the quality of the solution found.

5. EXPERIMENTAL EVALUATION

To evaluate the proposed methods we use synthetic and real-world social communication networks. We examine the running time of the algorithms, analyze the structural characteristics of the discovered subgraphs, and we present a case study. An implementation of the proposed methods and all the datasets used in our experimentation are publicly available.[2]

5.1. Synthetic data

We construct a network with several planted subgraphs. The objective is to measure how the algorithms behave with respect to the density of the planted subgraphs and the topology network. We model the topology network G and the planted subgraphs G'_i as Erdős-Rényi random graphs and vary their expected degrees. The time interval of the simulation is denoted by T. For every edge in G we choose uniformly at random a time stamp, indicating the time that the edge is active. The interactions of the edges of each G'_i occur in some short time periods T' with $|T'_i| \ll |T|$. The planted subgraphs G'_i are generated to be non-overlapping in nodes and are planted into non-overlapping time intervals.

We test the ability of our algorithms to discover planted subgraphs with two families of artificial datasets, *Synthetic1* and *Synthetic2*. The family *Synthetic1* is designed to test the quality of the discovered subgraphs as a function of the density of the topology

[2]https://github.com/polinapolina/dynamic_dense_subgraph

Table I. Characteristics of the synthetic datasets. $|V|$: number of nodes in the dataset; $d(H)$: expected density of the planted subgraph; $d(G)$: expected density of the background network; $|T|$: length of the whole time interval (in time units); B: time budget required to cover the subgraph activity (in time units); K: number of continuous time intervals that contain subgraph activity; $|C|$: number of planted subgraphs.

Name	$\|V\|$	$d(H)$	$d(G)$	$\|T\|$	B	K	$\|C\|$
Synthetic1	100	4	1–6	1000	100	3	3
Synthetic2	100	2–7	4	1000	100	3	3

Table II. Characteristics of real-world datasets. $|V|$: number of nodes; $|\pi(E)|$: number of edges of the topology network; $|E|$: number of interactions; $|T|$: time span of the dataset (in days); $d(\pi(G))$: density of the topology network; $d(H)$: density of the densest subgraph of the topology network.

Name	$\|V\|$	$\|\pi(E)\|$	$\|E\|$	$\|T\|$	$d(\pi(G))$	$d(H)$
Facebook	4117	5143	10000	104	2.49	5.29
Twitter	4605	6006	11868	93	2.60	10.11
Tumblr	1980	2454	7645	89	2.47	7.0
Students	889	2267	9837	120	5.10	11.29
Enron	1143	2019	6245	8080	3.53	14.38
FacebookL	45813	183412	10000000	104000	8.01	27.36
TwitterL	162207	324531	10000000	78362	4.00	21.24

network. On the other hand, the family *Synthetic2* consists of a topology network with fixed density, while the density of the planted subgraphs varies.

Both datasets span an interval T with $|T| = 1000$ time units and contain 3 non-overlapping planted subgraphs G'_1, G'_2 and G'_3. Each planted subgraph G'_i is covered by $K = 3$ time intervals with total length of $|T'_i| = 100$ time units. The expected density of the underlying network in *Synthetic1* varies from 1 to 6, while subgraphs are fixed to be 5-cliques. The expected density of the underlying network in *Synthetic2* is fixed to 4, while the density of the planted 8-node subgraphs varies from 2 to 7. A summary of the characteristics of the datasets is given in Table I.

5.2. Real-world data

We use seven real-world datasets. Five datasets are communication networks, where dynamic subgraphs are expected to be present. The other two datasets are larger in size, and we use them to test the scalability of our algorithms. The characteristics of these datasets are summarized in Table II.

Facebook [Viswanath et al. 2009]: This dataset is a 3-month subset of Facebook activity in a New Orleans regional community. The dataset contains anonymized list of wall posts (interactions). The subset covers time period from 9.05.06 to 20.08.06.

Twitter: The dataset tracks activity of Twitter users in Helsinki during 08.2010–10.2010. As interactions we consider tweets that contain mentions of other users.

Tumblr: A subset of the Memetracker dataset,[3] which contains quoting between Tumblr users. The subset covers three months: 02.2009–04.2009.

Students:[4] The activity log of a student online community at the University of California, Irvine. Nodes represent students and edges represent messages, where the message direction is suppressed. We use a subset of the dataset that covers four months of communication from 27.06.2004 to 26.10.2004.

[3]http://snap.stanford.edu/data/memetracker9.html
[4]http://toreopsahl.com/datasets/#online_social_network

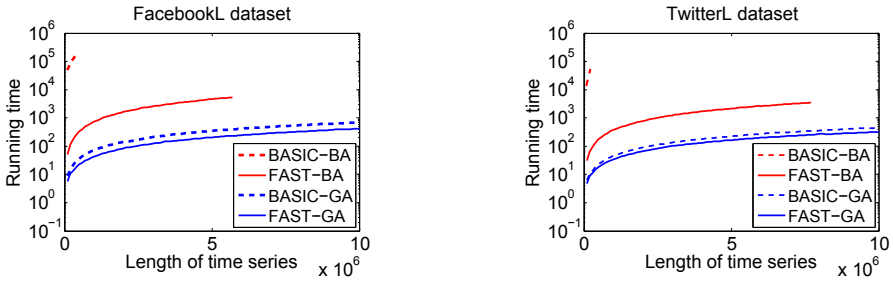

Fig. 1. Running time of the proposed algorithms (naïve implementation and pruned versions). x-axis: number of time units in input network; y-axis: running time in seconds.

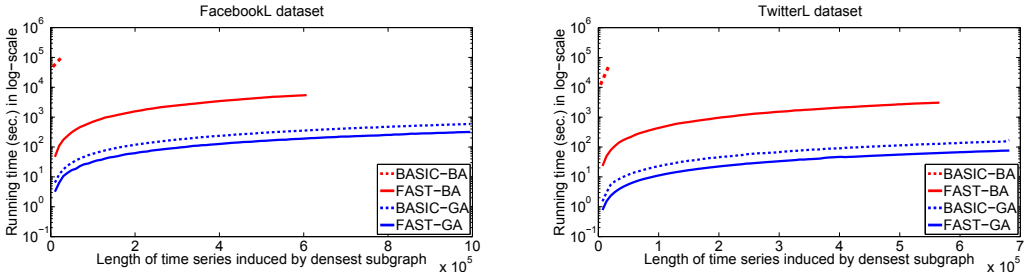

Fig. 2. Running time of one interval search iteration of basic and faster versions of the proposed algorithms.

Enron:[5] This is a well-known dataset that contains email communication of the senior management in a large company. It spans over 20 years from 1980.

FacebookL: This dataset is constructed by sequential concatenation of the *Facebook* dataset; it contains 10 million records.

TwitterL: Similar to *FacebookL*, this dataset is a concatenation of the *Twitter* dataset and contains 10 million records. The last two datasets are mainly used for scalability experiments.

5.3. Running time

We conduct our experiments on a machine with an 8-core Intel Xeon CPU 3.30 GHz and 15.6 GB of memory. We run one iteration of Algorithm 1 with $K = 1$ and $B = 1$ day on *FacebookL* and *TwitterL*. We vary the number of input time units from 100 K to 10 million. The total running time of the algorithm until convergence is shown in Figure 1. We observe that the algorithm scales very well with the input size.

The running time grows slowly with the size of the input data. However, the total running time depends heavily on the number of time units induced by the densest subgraph, as discovered in the first iteration of the algorithm. In Figure 2 we show the running time of the algorithm as a function of this number. We clearly see that our concave-based pruning techniques yield large speed-up. For BA the speed-up is quite dramatic. For GA, which is significantly faster than BA, even in its basic form, concave-based pruning improves the running time around an order of magnitude.

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

5.4. Discovering hidden structure

The next step is to evaluate the ability of our methods to discover dynamic dense subgraphs. We start from the synthetic datasets, where the ground-truth subgraphs are known, and we proceed to real-world data. We examine the sensitivity to initialization, the effects of the budget parameters, and the characteristics of the discovered subgraphs.

Planted subgraphs. We test how well our algorithms detect the planted subgraphs for different levels of background noise and subgraph density. We quantify the quality of our algorithms by measuring *precision* and *recall*, with respect to the ground-truth subgraphs. We also report the F-measure, the harmonic mean of precision and recall. Reported results are averages over $1\,000$ independent runs.

Aiming to retrieve all three planted subgraphs we run the algorithms three times, removing the edges of the discovered subgraph after each iteration. The results of this experiment, for the two families of synthetic datasets, and for algorithms GA and BA, are shown in Figure 3. Precision, recall, and F-measure are computed after matching the three discovered subgraphs with the three ground-truth subgraphs.

Recall that *Synthetic1* contains three subgraphs based on a 5-clique. Both algorithms are able to discover these subgraphs correctly when the average degree of the underlying graph is smaller than the average degree of the planted subgraphs. Even when the subgraph density is equal to the background network density (around 4), the algorithms achieve high accuracy. Precision and recall degrade at the same rate, indicating that with the increase of the background network density the algorithms retrieve less nodes from the planted subgraphs as well as nodes outside the subgraphs. Nevertheless, the measures do not drop too much, implying that the 5-clique spanning 3 short time intervals is distinguishable even within a dense background network.

The results on the *Synthetic2* dataset are similar: both algorithms perform well when the density of the background network is smaller than the density of the planted subgraph.

Effect of random seeds. Both of our algorithms are instances of Algorithm 1, and they require a time interval set as an initial seed. In the previous experiments we initialize the interval seed $\mathcal{T}_0$ with the whole time interval T spanned by the dataset. Starting from $\mathcal{T}_0 = \{T\}$ ensures that the subgraph we discover belongs to some dense structure in the topology network. However, if such a dense structure occurs in a scattered manner, the initialization $\mathcal{T}_0 = \{T\}$ may mislead. To overcome this problem and avoid dense structures that cannot be covered in the given time budget, we initialize Algorithm 1 with many random time intervals, and return the best solution found.

The effect of random initializations is shown in Figure 4. The experiments are shown for *Tumblr* and *Students*. The figures show the best density discovered by our algorithms, with $1\,000$ independent random restarts. As expected, random initializations improve the performance of the algorithms. The most significant improvement is obtained for the *Students* dataset. We also experiment with a baseline that finds the densest subgraph over *all* possible intervals that satisfy the time budget B (no iterative process is followed). We see that our algorithms perform significantly better than this baseline.

Discovered subgraphs. Table III reports the densities of the subgraphs discovered by our algorithms in real-world datasets. We use 200 random initializations. Here and in later experiments we compare our algorithms with the baseline that finds the densest subgraph over all intervals that satisfy the time budget B.

Overall, we observe that GA and BA perform equally well, while in some settings BA yields denser subgraphs than GA.

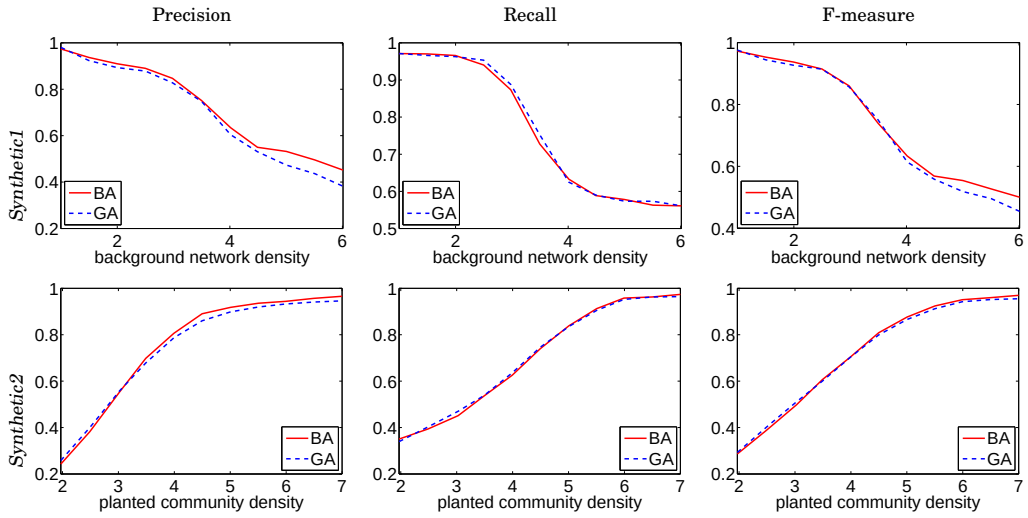

Fig. 3. Precision, recall, and F-measure on *Synthetic1* and *Synthetic2*. Each row corresponds to dataset family. For *Synthetic1* the planted subgraph is a 5-clique, and the x-axis represents the background network density. For *Synthetic2* the background network density is set to 4, and the x-axis represents the density of a planted subgraph of 8 nodes. The number is planted subgraphs is 3. The activity of the planted subgraphs is spread over 3 intervals.

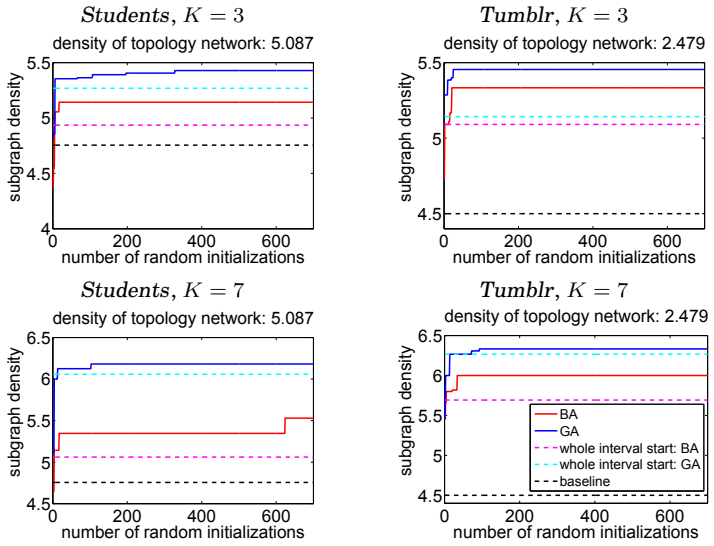

Fig. 4. Effect of random initializations on the real-world datasets. Time budget B is set to 7 days.

For a fixed value of the time budget B, the density of the discovered subgraph increases with K. For small values of K (1 to 3), the density of the subgraphs discovered by our algorithm is equal or close to the density of the subgraphs discovered by the baseline. This behavior is expected, as the baseline examines all possible intervals, while our algorithms use only some random intervals for initialization. However, as the value of K increases, the algorithms take advantage of the provided flexibility to

Table III. Densities of discovered subgraphs. B denotes time budget in days.

			Community density			Community size		
Dataset	B	K	GA	BA	BASE	GA	BA	BASE
Facebook	1	1	2.4	2.4	2.4	5	5	5
		5	3.66	3.66	2.4	6	6	5
		10	3.75	3.75	2.4	8	8	5
	7	1	3	3	3	6	6	6
		5	3.87	4	3	16	9	6
		10	4.28	4.47	3	14	17	6
Twitter	1	1	4	4	4	6	6	6
		5	5.11	5.33	4	9	9	6
		10	6.4	6.4	4	10	10	6
	7	1	4.66	4.66	4.66	9	9	9
		5	6	6.22	4.66	14	9	9
		10	6.92	7.2	4.66	13	15	9
Tumblr	1	1	3.86	3.86	3.86	30	30	30
		5	5.11	5.25	3.86	9	8	30
		10	5.81	6.18	3.86	11	11	30
	7	1	4.5	4.5	4.5	8	8	8
		5	5.88	6	4.5	18	11	8
		10	6.71	6.8	4.5	14	15	8
Students	1	1	3.41	3.33	3.42	17	15	21
		5	4.66	4.62	3.42	9	16	21
		10	5.5	5.62	3.42	16	16	21
	7	1	4.75	4.69	4.75	45	43	45
		5	5.82	6	4.75	46	25	45
		10	6.76	7.12	4.75	34	41	45
Enron	1	1	6.18	6.18	6.18	11	11	11
		5	10	10.37	6.18	17	16	11
		10	12.2	12.38	6.18	20	21	11
	7	1	6.36	6.36	6.36	11	11	11
		5	11.26	11.23	6.36	19	26	11
		10	13.07	13.07	6.36	28	28	11

use many intervals effectively; for $K > 3$ both algorithms always outperform the baseline.

Furthermore, as we can see by contrasting Tables II and III, the discovered subgraphs are almost as dense as the densest subgraphs on the whole topology network, even though the time budget is significantly smaller than the time span of the dataset. For example, the densest subgraph of the over 20-year-long *Enron* dataset has average degree 14.38, while we were able to discover a subgraph with average degree 13.07 in a budget of 7 days, spanning 10 time intervals.

A typical retrieved dense subgraph has a size of about 10 to 20 nodes, which is reasonably small for a dense social community, such as friends or colleagues. Moreover, this value is independent from the choice of parameters, and is similar for all the datasets. In other words, for the datasets we experiment with, the retrieved subgraphs represent tightly-connected and small subgraphs, while, the subgraphs found as the densest subgraphs of the underlying network (Table II) have large size (up to 200 nodes) and large variance.

Retrieved intervals. In this section we present our results with respect to the time intervals that cover the discovered subgraphs. All the results in that section were obtained without use of random initialization.

Table IV. Total time span of the discovered subgraphs. B: time budget (days); actual B: time budget actually used (days); span: length (in days) of the minimal time interval needed to cover the retrieved subgraph in one time interval.

			GA		BA	
Dataset	B	K	actual B	span	actual B	span
Facebook	1	3	0.39	9.09	0.79	18.1
		7	0.72	20.2	0.72	29.5
	7	3	4.44	20.2	3.57	20.2
		7	5.61	73	4.9	90.1
Twitter	1	3	0.41	32.3	0.46	51.7
		7	0.69	32.7	0.92	72.4
	7	3	6.99	32.7	5.84	79.3
		7	6.49	79.5	5.09	79.3
Tumblr	1	3	0.87	33.4	0.77	56.8
		7	0.82	49.4	0.9	49.4
	7	3	6.76	44.9	5.92	37.3
		7	6.99	82.1	5.92	60.4
Students	1	3	0.75	8.51	0.54	8.61
		7	0.99	29.5	0.92	55.3
	7	3	6.91	25.3	6.66	32.2
		7	6.56	76.1	6.96	73.4
Enron	1	3	0.83	69.4	0.83	69.4
		7	0.98	320	0.88	158
	7	3	4.32	69.4	6.45	158
		7	6.64	320	5.35	320

As shown in Table III, GA and BA find a significantly dense subgraph, comparable to the densest subgraph of the underlying network. Table IV illustrates that the discovered subgraphs must exploit the budget of several intervals K to fit the time budget B. Column B reports the total length of the retrieved time intervals. Column $span$ indicates the minimal length of a *single* time interval that covers all the edges of the retrieved subgraph. According to Table IV, the coverage of a retrieved dense subgraph by one time interval requires a large time budget (up to several months), which is comparable to the whole time span of the dataset. Moreover, $span$ is typically much larger than the time budget we spent constructing the K intervals.

Both GA and BA use the time budget extensively, while GA tends to use B more tightly.

Table V demonstrates how the density of the resulting subgraph is distributed among the intervals. On average, each time interval covers a subgraph of a small density, while the union of the intervals results in a denser structure. The density of the densest subgraph is significantly lower than the density of the subgraph in the union of all K intervals.

An example of a discovered subgraph in the *Facebook* dataset, along with its corresponding time intervals, is illustrated in Figure 5. Each subplot corresponds to one time interval with duration shown in the figure caption, while the last subplot shows all the interactions in the union of all time intervals. The active edges of the time interval are marked in red. The example illustrates the case discussed above: a dense subgraph is composed by small components scattered among the time intervals of various time lengths, with none of these small components being dense enough.

5.5. Discovering multiple subgraphs

We discussed previously how to adapt the proposed algorithms in order to discover multiple subgraphs on the synthetic datasets (Figure 3). We use the same strategy for

Table V. Density of the induced subgraphs of the retrieved intervals. Time budget B is fixed to 3 days; K: number of intervals; $d(\mathcal{T})$: density of the discovered subgraph in the union of K intervals; avg $d(T_i)$ and max $d(T_i)$: average and maximal density of the subgraph, covered by one of K time intervals.

		$d(\mathcal{T})$		avg $d(T_i)$		max $d(T_i)$	
Dataset	K	GA	BA	GA	BA	GA	BA
Facebook	3	3.33	3.66	1.53	1.74	2	2.4
	5	3.75	3.75	1.45	1.52	2	2.33
	10	4	4.15	1.27	1.29	1.5	1.71
Twitter	3	4.66	4.57	2.66	2.16	4	4
	5	5.8	5.77	2.11	1.92	3.71	4
	10	6.4	6.4	1.61	1.6	3.71	3.71
Tumblr	3	5.27	5.23	2.41	2.52	3.2	3.33
	5	5.63	5.71	2.05	2.06	3	3.38
	10	6.18	6.57	1.85	1.83	2.88	3.33
Students	3	4.43	4.72	2.52	2.14	3.04	3.4
	5	5	5.4	1.6	1.7	2	2.66
	10	6.08	6.13	1.7	1.63	2	2.42
Enron	3	9.6	9.6	4.14	4.14	5.09	5.09
	5	10.8	10.8	3.44	3.47	4.6	5.09
	10	12.7	12.7	2.62	2.9	4.54	5.16

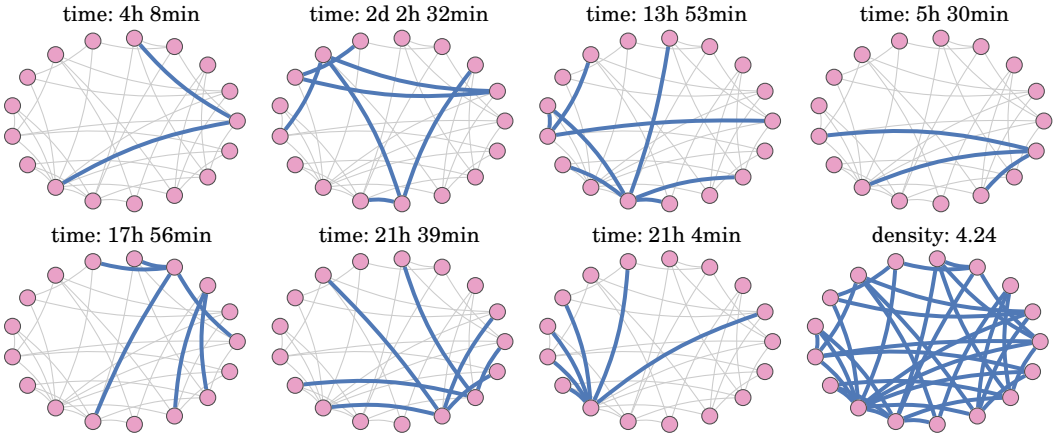

Fig. 5. Example of subgraph found by GA. *Facebook* dataset, $K = 7$ and $B = 7$ days.

the real-world datasets: we run the subgraph-finding algorithms $n = 20$ times to obtain n subgraphs; after each iteration we delete the edges of the discovered subgraph.

Let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be the set of n dynamic subgraphs discovered by this iterative strategy. We can apply the same strategy to discover a set of n dense subgraphs on the underlying topology network. Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be the set of those dense subgraphs. We compare the two sets of subgraphs, $\mathcal{D}$ and $\mathcal{C}$. The results are shown in Figure 6.

We that the baseline collection $\mathcal{C}$ contains some subgraphs that are very dense, however, the density decreases rapidly. One explanation for this behavior is that the first subgraphs discovered in $\mathcal{C}$ are large in size, containing hundreds or thousands of nodes, and their removal decreases the density of the network rapidly. On the other hand, the proposed algorithms, GA and BA, produce a more balanced set of subgraphs, both in size and in density. Furthermore, the activity of the subgraphs in $\mathcal{C}$ is scattered in

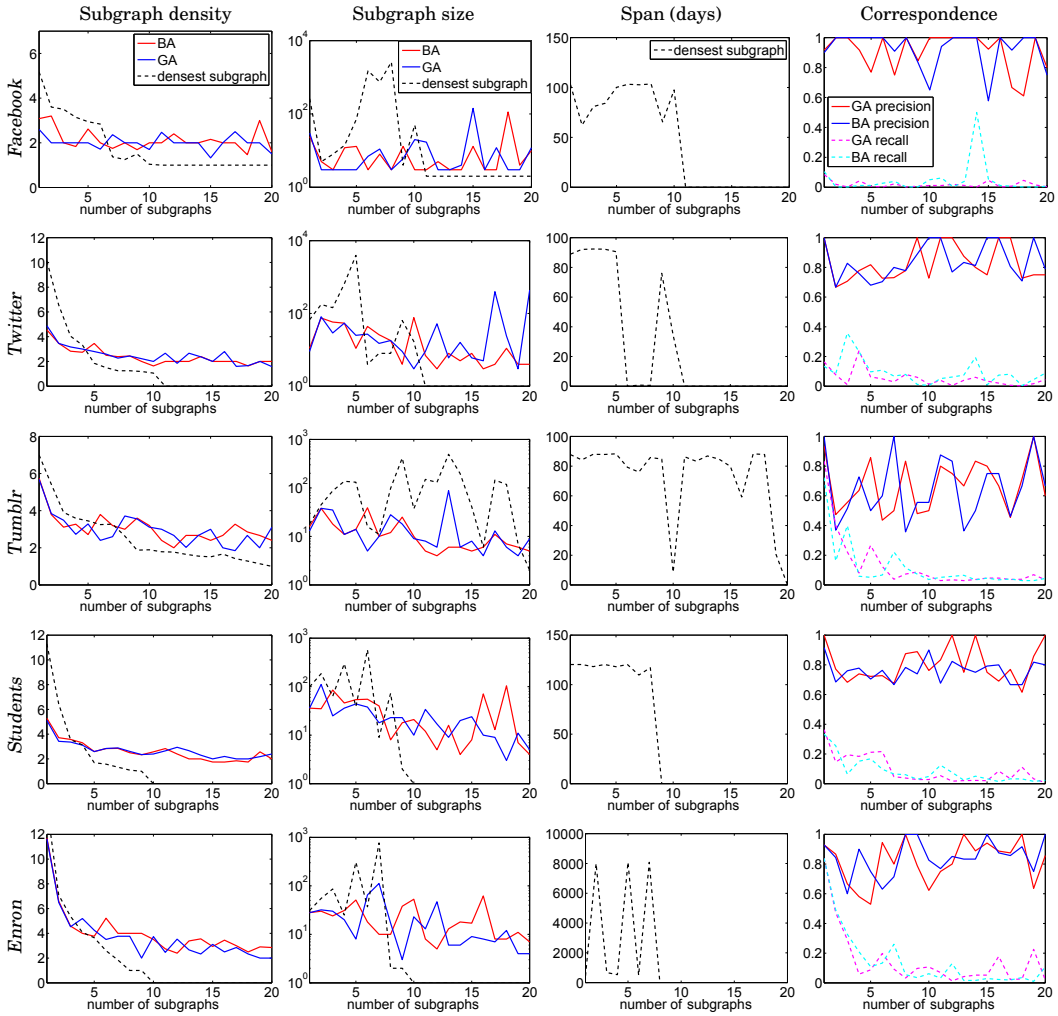

Fig. 6. Comparison of multiple discovered subgraphs with multiple search for dense subgraph of topology network. Each row represents a dataset. The first and second columns show the density and the size of multiple discovered subgraphs. The third column shows the time span of multiple subgraphs discovered on the underlying network. The fourth column shows the relations between multiple output subgraphs and the densest subgraphs of the topology network. All runs of the dynamic subgraph-finding method were performed with $K = 7$ and $B = 3$ days.

time, and the time span of a single interval needed to cover those subgraphs is large (third row in Figure 6).

We test whether the dynamic subgraphs $\mathcal{D}$, discovered by GA and BA, are subsets of the static subgraphs $\mathcal{C}$. To test this hypothesis, we compare every discovered subgraph D_i to every densest subgraph C_j, $1 \leq i, j \leq n$, and find the best match. The best match for subgraph D_i is defined as $M(D_i) \in \mathcal{C}$, such that

$$M(D_i) = \arg\max_{C_j} P(D_i, C_j),$$

Table VI. Characteristics of the subgraphs discovered by GA and the densest-subgraph algorithm on the *Twitter* dataset. The hashtags are annotated by their frequency and ones that occur in both subgraphs are marked by bold font.

Method	Size	Density	Hashtags
GA	9	4.9	***aaltoes*: 52**, ***startup*: 12**, ***vc*: 11**, ***summerofstartups*: 11**, ***entrepreneur*: 7**, ***startups*: 4**, ***web*: 3**, ***slush10*: 2**, ***skype*: 2**, ***funrank*: 2**, ***africa*: 2**, ***mobile*: 2**, ***demoday*: 2**, ***design*: 2**, ***linkedin*: 2**, ***aalto*: 2**
Densest	67	10.1	***aaltoes*: 80**, ***summerofstartups*: 28**, ***startup*: 18**, ***vc*: 13**, *ff*: 11, *fb*: 8, *elonmerkki*: 7, ***entrepreneur*: 7**, ***slush10*: 6**, *newtwitter*: 5, *2010mvv*: 4, *garage48*: 4, *facebook*: 4, ***web*: 4**, ***startups*: 4**, *smss2010*: 4, *mvv2010*: 3, *angrybirds*: 3, *fail*: 3, *spotonloc*: 3, *fif2010*: 3, *baltic*: 3, ***africa*: 3**, ***mobile*: 3**, ***demoday*: 3**, *helsinki*: 3, *gov20*: 3, *e20*: 3, *failcon*: 2, *bacon*: 2, *mindtrek*: 2, ***skype*: 2**, ***funrank*: 2**, *nxcfi*: 2, *blog*: 2, *yc*: 2, *hankenes*: 2, ***design*: 2**, *education*: 2, ***linkedin*: 2**, *nokia*: 2, *n8*: 2, ***aalto*: 2**

where P measures precision with C_j as ground truth. Using the best matching subgraph $M(D_i)$ we calculate precision $I_P(D_i) = P(D_i, M(D_i))$ and recall $I_R(D_i) = R(D_i, M(D_i))$ for each discovered subgraph. The resulting plots are shown in the last row of Figure 6. In these plots we refer to I_P and I_R as precision and recall correspondingly. High precision I_P and low recall I_R indicate that the subgraphs $C_1, \ldots, C_n$ are supersets of the subgraphs, obtained by GA and BA. We conclude that the proposed algorithms are able to uncover subgraphs hidden in the dense subgraphs of the underlying network.

5.6. *Twitter* case study

We conduct a case study with the *Twitter* dataset, which in addition to interactions between users it also contains the tweet texts. To evaluate the coherence of the discovered subgraphs we inspect the set of hashtags that appear in the communication among the members of the retrieved subgraph. For each subgraph we select the most frequent hashtags.

As an example, the GA algorithm with $K = 7$ intervals and budget $B = 3$ days discovers a subgraph C_1 of 9 Twitter users, which include organizations, such as Aalto Entrepreneurship Society (*@aaltoes*) and Aalto Venture Garage (*@AaltoGarage*). The density of C_1 is 4.9, and the most popular hashtags appearing in C_1 are shown in Table VI.

For comparison, the densest subgraph of the underlying network is a subgraph C_2, consisting of 67 nodes, and it is a superset of C_1. The density of C_2 is 10.1, and the minimal time interval that covers this subgraph is 89 days; much longer than the 3 days that C_1 spans. The number of hashtags used in C_2 is 112, and as can be seen in Table VI it is a less focused set. Furthermore, the hashtags of C_1 contain some of the most frequent hashtags in C_2. Overall, C_1 is a dense subgraph of C_2, which contains some of the most frequent hashtags, and whose members are interacting in a short time.

6. RELATED WORK

Community detection is one of the most studied problems in social-network analysis. A lot of research has been devoted to the case of static graphs, and the typical setting is to partitioning a graph into disjoint communities [Girvan and Newman 2002; Flake et al. 2000; Pons and Latapy 2006; van Dongen 2000]; a thorough survey on such methods has been compiled by Fortunato [2010].

Typically the term “dynamic graphs” refers to the model where edges are added or deleted. In this setting, once an edge is inserted in the graph it stays “alive” until the

current time or until it is deleted. For example, this setting is used to model the process in which individuals establish friendship connections in a social network. On the contrary, our model intends to capture the continuous interaction between individuals. In the dynamic-graph setting, researchers have studied models that capture arrival of new nodes and edges in the network [Kumar et al. 2006; Leskovec et al. 2008; Zhou et al. 2014; Myers and Leskovec 2014] and the process of how groups and communities are formed [Backstrom et al. 2006]. Additionally, in the context of dynamic graphs, different algorithms have been proposed for maintaining records of graph characteristics [Li et al. 2014] and for mining rules for graph evolution [Berlingerio et al. 2009].

With respect to community detection in time-evolving graphs, the prominent line of work is to consider different graph snapshots, find communities in each snapshot separately (or by incorporating information from previous snapshots), and then establish correspondences among the communities in consecutive snapshots, so that it is possible to study how communities *appear*, *disappear*, *split*, *merge*, or *evolve*. A number of research papers follows this framework [Asur et al. 2009; Greene et al. 2010; Lin et al. 2008; Palla et al. 2007; Sun et al. 2007]. Similar recent works apply concepts of Laplacian dynamics [Mucha et al. 2010] and frequent pattern mining [Berlingerio et al. 2013] to ensure coherence and sufficiency of communities found in sequence of graph snapshots.

We should point out that our approach is not directly comparable to methods for community detection, in the classic sense. Instead our approach should be viewed as a method for finding dense subgraphs in dynamic networks. For example, a common way to formulate the community-detection problem is to seek to maximize the modularity measure [Newman 2006]. Classic modularity is defined over a *partitioning* of the graph nodes, while we focus on finding a *single* dense community in the temporal network (although we also experiment with finding top-k overlapping densest subgraphs). We should also note that adapting and optimizing modularity-based measures with time constraints is a highly non-trivial task; for example, increasing the size of an interval may actually decrease modularity. Overall, contrasted with modularity-based methods, our approach uses a different objective function, it satisfies different constraints, and even has different output.

Many dynamic-graph studies are dedicated to the event-detection problem. The comprehensive tutorial by Akoglu et al. [2014] covers recent research on this topic. The majority of the work focuses on how to compare different graph snapshots, and it aims to detect those snapshots that the graph structure changes significantly. The research tools developed in this area include novel metrics for graph similarity [Papadimitriou et al. 2010] and graph distance—see the survey of Gao et al. [2010] and the recent paper by Sricharan and Das [2014]—as well as extending scan-statistics methods for graphs [Priebe et al. 2005], while a number of papers relies on matrix-decomposition methods [Akoglu and Faloutsos 2010; Ide and Kashima 2004]. The main challenge addressed in this paper, in contrast to other event-detection problems, is that we aim to discover simultaneously the subgraph of the event and the relevant time intervals. Should the subgraph be known, a wide range of anomaly-detection methods could be used [Chandola et al. 2009].

Similarly to dynamic-graph event detection, constructing a static graph (or a sequence of static graphs) with incorporated temporal information is a common approach for dynamic-graph problems, best reviewed in an extensive survey by Holme and Saramäki [2012]. An interesting example of embedding temporal data into a static graph is a dynamic clustering approach from Rosvall et al. [2014]. They use historical temporal data to learn probabilities of 2-hop-paths to produce clustering. However, the discovered clustering is global in time, there are no associated burst time intervals detected, and no constraints on time. Another example is a spatio-temporal event de-

tection method by Rozenshtein et al. [2014]. The goal of the paper is to find a subset of nodes that are close to each other and have high activity levels. The activity score of a node is inferred from temporal records and depends on a given time interval. However, the data is viewed as sequence of network snapshots, which are precessed independently.

To our knowledge, the approach that is best aligned with our problem setting, is presented by Bogdanov et al. [2011], for the problem of mining heavy subgraphs in time-evolving networks. Yet, there are important differences. First, the approach of Bogdanov et al. is still based on network snapshots, and thus sensitive to boundary-quantization effects. Second, their concept of heavy subgraphs is based on edge weights, and their discovery problem maps to *prize collecting Steiner tree*, as opposed to a density-based objective.

Hu et al. [2005] propose a framework for mining frequent coherent dense subgraphs across a sequence of biological networks. Their core concept is to construct a second-order graph, which represents co-activity of edges in the initial graph. As with the previous papers, Hu et al. work with network snapshots, which is quite a different model than the one we consider in this paper.

The goal of this work is to discover dense subgraphs that occur in a *short* period of time. If the actual nodes of the subgraph are already known, and we are interested in finding the intervals in which the graph is the densest, then the problem can be viewed as a burst detection problem. Modeling and discovering bursts is a well-studied topic. A well-known approach by Kleinberg [2003] models the delays between the events with an exponential model, and uses a dynamic program to discover hierarchical bursts. An alternative approach is to count events in a window of predetermined length: for example, Ihler et al. [2006] models event counts with Poisson process, while Fung et al. [2005] uses Binomial distribution. Alternative discovery techniques include wavelet analysis [Zhu and Shasha 2003], Fourier analysis [Vlachos et al. 2004], and notions inspired by Mechanics [He and Parker 2010]. The obvious difference between these techniques and our approach is that not only we need to find the intervals but also the subgraph as well. Moreover, note that the aforementioned methods have their own objective functions, and so cannot be used to optimize the problem defined here. Nevertheless, an interesting direction is to see whether the modeling of intervals techniques can be combined in a meaningful way with discovering dense subgraphs.

In summary, in contrast to the existing work, we introduce a new point-of-view in the area of dynamic graphs, namely, we incorporate in our analysis point-wise interactions between the network nodes.

7. CONCLUDING REMARKS

In this work we considered the problem of analyzing interaction networks and finding dynamic dense subgraphs. We formulated this structure-discovery problem by asking to find a dense subgraph whose edges occur in short time intervals. We proved that the problem is **NP**-hard, and provided efficient algorithms. We demonstrated how to speed up the proposed algorithms by taking advantage of the concavity property of the objective function and by using fractional-programming techniques.

Our work is a step towards a more refined analysis of social networks, in which interaction information is taken into account, and it is used to provide a more accurate description of communities and their dynamics in the network.

This work opens many possibilities for future research. In particular, it will be interesting to enhance the definition of dynamic dense structure by incorporating additional information, either text and attributes of the network nodes, or information regarding the frequency and the statistical properties of the network interactions.

REFERENCES

Alok Aggarwal, MariaM. Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. 1987. Geometric applications of a matrix-searching algorithm. *Algorithmica* 2, 1–4 (1987), 195–208.

Leman Akoglu and Christos Faloutsos. 2010. Event detection in time series of mobile communication graphs. In *Army Science Conference*.

Leman Akoglu, Hanghang Tong, and Danai Koutra. 2014. Graph-based anomaly detection and description: A survey. *Data Mining and Knowledge Discovery* 28, 4 (2014).

Albert Angel, Nikos Sarkas, Nick Koudas, and Divesh Srivastava. 2012. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proceedings of the VLDB Endowment* 5, 6 (2012), 574–585.

Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. 2000. Greedily finding a dense subgraph. *Journal of Algorithms* 34, 2 (2000).

Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. 2009. An event-based framework for characterizing the evolutionary behavior of interaction graphs. *TKDD* 3, 4 (2009).

Lars Backstrom, Daniel P. Huttenlocher, Jon M. Kleinberg, and Xiangyang Lan. 2006. Group formation in large social networks: membership, growth, and evolution. In *KDD*.

Gary D Bader and Christopher WV Hogue. 2003. An automated method for finding molecular complexes in large protein interaction networks. *BMC bioinformatics* 4, 1 (2003), 1.

Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. 2009. Mining Graph Evolution Rules. In *ECML PKDD*.

Michele Berlingerio, Fabio Pinelli, and Francesco Calabrese. 2013. ABACUS: frequent pAttern mining-BAsed Community discovery in mUltidimensional networkS. *DMKD* 27, 3 (2013), 294–320.

Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*. 119–130.

Petko Bogdanov, Misael Mongiovì, and Ambuj K. Singh. 2011. Mining Heavy Subgraphs in Time-Evolving Networks. In *ICDM*.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 15.

Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*.

Werner Dinkelbach. 1967. On Nonlinear Fractional Programming. *Management Science* 13, 7 (1967), 492–498.

Uriel Feige. 2004. Approximating Maximum Clique by Removing Subgraphs. *SIAM J. Discrete Math.* 18, 2 (2004), 219–225.

Gary William Flake, Steve Lawrence, and C. Lee Giles. 2000. Efficient identification of Web communities. In *KDD*.

Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486 (2010).

Gabriel Pui Cheong Fung, Jeffrey Xu Yu, Philip S. Yu, and Hongjun Lu. 2005. Parameter Free Bursty Events Detection in Text Streams. In *VLDB*. 181–192.

Zvi Galil and Kunsoo Park. 1992. Dynamic Programming with Convexity, Concavity, and Sparsity. *Theoretical Computer Science* 92, 1 (1992), 49–76.

Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. 2010. A survey of graph edit distance. *Pattern Analysis and applications* 13, 1 (2010).

David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st International Conference on Very Large Databases*. 721–732.

M. Girvan and M. E. J. Newman. 2002. Community structure in social and biological networks. *PNAS* 99 (2002).

David F. Gleich and C. Seshadhri. 2012. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD*.

Andrew V Goldberg. 1984. Finding a maximum density subgraph. *University of California Berkeley Technical report* (1984).

Derek Greene, Donal Doyle, and Padraig Cunningham. 2010. Tracking the Evolution of Communities in Dynamic Social Networks. In *ASONAM*.

Dan He and D. Stott Parker. 2010. Topic Dynamics: An Alternative Model of Bursts in Streams of Topics. In *KDD*. 10.

Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics reports* 519, 3 (2012), 97–125.

Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou. 2005. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics* (2005).

Tsuyoshi Ide and Hisashi Kashima. 2004. Eigenspace-based anomaly detection in computer systems. In *KDD*.

Alexander Ihler, Jon Hutchins, and Padhraic Smyth. 2006. Adaptive Event Detection with Time-varying Poisson Processes. In *KDD*. 207–216.

Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. 1999. The Budgeted Maximum Coverage Problem. *IPL* 70, 1 (1999), 39–45.

Jon Kleinberg. 2003. Bursty and Hierarchical Structure in Streams. *DMKD* 7, 4 (2003), 373–397.

Ariel Kulik, Hadas Shachnai, and Tami Tamir. 2009. Maximizing Submodular Set Functions Subject to Multiple Linear Constraints. In *SODA*.

Ravi Kumar, Jasmine Novak, and Andrew Tomkins. 2006. Structure and evolution of online social networks. In *KDD*.

Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. 2008. Microscopic evolution of social networks. In *KDD*.

Jure Leskovec, Kevin J. Lang, and Michael W. Mahoney. 2010. Empirical comparison of algorithms for network community detection. In *WWW*.

Rong-Hua Li, Jeffrey Xu Yu, and Rui Mao. 2014. Efficient Core Maintenance in Large Dynamic Graphs. *TKDE* 26, 10 (2014), 2453–2465.

Yu Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle Tseng. 2008. Facetnet: A Framework for Analyzing Communities and Their Evolutions in Dynamic Networks. In *WWW*.

Peter J Mucha, Thomas Richardson, Kevin Macon, Mason A Porter, and Jukka-Pekka Onnela. 2010. Community structure in time-dependent, multiscale, and multiplex networks. *science* 328, 5980 (2010), 876–878.

Seth A. Myers and Jure Leskovec. 2014. The Bursty Dynamics of the Twitter Information Network. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14)*. 12.

Mark EJ Newman. 2006. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences* 103, 23 (2006), 8577–8582.

G. Palla, A. Barabási, and T. Vicsek. 2007. Quantifying social group evolution. *Nature* 446 (2007).

Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. 2010. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications* 1, 1 (2010).

Pascal Pons and Matthieu Latapy. 2006. Computing Communities in Large Networks Using Random Walks. *Journal of Graph Algorithms Applications* 10, 2 (2006).
Carey E Priebe, John M Conroy, David J Marchette, and Youngser Park. 2005. Scan statistics on enron graphs. *Computational & Mathematical Organization Theory* 11, 3 (2005).
Martin Rosvall, Alcides V Esquivel, Andrea Lancichinetti, Jevin D West, and Renaud Lambiotte. 2014. Memory in network flows and its effects on spreading dynamics and community detection. *Nature communications* 5 (2014).
Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. 2014. Event detection in activity networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1176–1185.
Roded Sharan and Ron Shamir. 2000. CLICK: a clustering algorithm with applications to gene expression analysis. In *Proceedings of the International Conference of Intelligent Systems for Molecular Biology (ISMB)*, Vol. 8. 16.
Kumar Sricharan and Kamalika Das. 2014. Localizing anomalous changes in time-evolving graphs. In *SIGMOD*.
Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. 2007. GraphScope: parameter-free mining of large time-evolving graphs. In *KDD*.
Stijn van Dongen. 2000. *Graph Clustering by Flow Simulation*. Ph.D. Dissertation. University of Utrecht.
Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. 2009. On the Evolution of User Interaction in Facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*. Barcelona, Spain.
Michail Vlachos, Christopher Meek, Zografoula Vagena, and Dimitrios Gunopulos. 2004. Identifying Similarities, Periodicities and Bursts for Online Search Queries. In *SIGMOD*. 131–142.
Robert E. Wilber. 1988. The Concave Least-Weight Subsequence Problem Revisited. *J. Algorithms* 9, 3 (1988), 418–425.
Rui Zhou, Chengfei Liu, Jeffrey Xu Yu, Weifa Liang, and Yanchun Zhang. 2014. Efficient Truss Maintenance in Evolving Networks. *arXiv preprint arXiv:1402.2807* (2014).
Yunyue Zhu and Dennis Shasha. 2003. Efficient Elastic Burst Detection in Data Streams. In *KDD*. 336–345.

A. PROOFS

PROOF OF PROPOSITION 3.1. We will show that the decision version of Problem 3.1 is **NP**-complete.

We are given an interaction network G, budgets K, B, and a threshold σ, and we need to answer whether there is a node set W and a time-interval set $\mathcal{T}$, which satisfy the two budget constraints, and for which $q(W, \mathcal{T}; G) \geq \sigma$.

The problem is clearly in **NP**. To prove the hardness, we obtain a reduction from VERTEXCOVER. An instance of VERTEXCOVER specifies a graph H and budget ℓ, and asks whether there is a set $V' \subseteq V$, such that $|V'| \leq \ell$ and each edge of the graph is adjacent to at least one of the nodes of V'.

Consider graph $H = (U, F)$ with n nodes and m edges, and budget ℓ. Let us define an interaction network $G = (V, E)$. The node set V consists of U and $n + 1$ additional auxiliary nodes, and the set of edges E is defined as follows: First we consider $n + 1$ distinct time points $t_0, \ldots, t_n$. At t_0 we add interactions between all the auxiliary nodes, and between auxiliary nodes and each $v \in U$. We arbitrarily order the nodes in U and let v_i be the i-th node. At time t_i we connect v_i with all its neighbors in H.

Assume a solution W and $\mathcal{T}$, for Problem 3.1, with budgets $K = \ell + 1$ and $B = 0$. We claim that W will contain all nodes and $\mathcal{T}$ will contain t_0 and the time points corresponding to the vertex cover of H.

Let us first prove that $W = V$ and $(t_0, t_0) \in \mathcal{T}$. Assume first that $(t_0, t_0) \notin \mathcal{T}$. Then, since the remaining time intervals have only edges between U, there must be at most $n(n-1)/2$ edges, yielding density at most $n - 1$. Let us replace one of the selected time intervals with t_0 and reset W to be auxiliary nodes. This solution gives us a density of n, which is a contradiction.

Now we have established that t_0 is a part of $\mathcal{T}$. A straightforward calculation shows that it is always beneficial to add auxiliary nodes to W, if they are not part of a solution. Once this is shown, we can show further that adding any missing nodes from U also improves the density. Consequently, $W = V$.

Set $\sigma = 2(n(n+1)/2 + n(n+1) + m)/(2n+1)$. The first two terms in the numerator correspond to the edges at t_0. The remaining m edges must come from the remaining time intervals. This is only possible if and only if the time intervals contain all edges from H, that is, the corresponding nodes cover every edge, which completes the reduction. □

PROOF OF PROPOSITION 4.1. (i) The proof is rather straightforward as Problems 3.1 and 4.2 have the same objective functions (up to fixed component W) and sets of constraints.

Let $(W, \mathcal{T})$ be an optimal solution to Problem 3.1 for a given interaction network G. Suppose $\mathcal{T}$ is not an optimal solution to Problem 4.2 given G and W. That means that there exits $\mathcal{T}^*$, such that $q(W, \mathcal{T}^*; G) > q(W, \mathcal{T}; G)$ with $|\mathcal{T}^*| \leq K$ and $\mathrm{span}(\mathcal{T}^*) \leq B$. That contradicts to our assumption that $(W, \mathcal{T})$ is an optimal solution to Problem 3.1. Consequently, $\mathcal{T}$ is an optimal solution for Problem 4.2 given W and G.

(ii) Analogous to (i). □

PROOF OF PROPOSITION 4.2. The proposition follows immediately from Lemma A.1 □

PROOF OF PROPOSITION 4.3. Let $x = |W|$ and $y = |E(\pi(G(W, \mathcal{T})))|$. Since $2y/x \leq B \cdot \alpha$, we must have

$$0 \leq Q_\alpha(\mathcal{T}) = 2\frac{y}{x} - \alpha \cdot \mathrm{span}(\mathcal{T}) \leq B \cdot \alpha - \alpha \cdot \mathrm{span}(\mathcal{T}),$$

which proves that $\mathrm{span}(\mathcal{T}) \leq B$. □

In order to prove several of the next propositions we need the following technical lemma.

LEMMA A.1. *Consider two functions $q : X \to \mathbb{R}$ and $c : X \to \mathbb{R}$. Define*

$$f(\alpha) = \arg\max_{x \in X} q(x) - \alpha c(x) \quad .$$

Then $q(f(\alpha))$ and $c(f(\alpha))$ decrease as α increases. Moreover, $q(f(\alpha_1)) = q(f(\alpha_2))$ if and only if $c(f(\alpha_1)) = c(f(\alpha_2))$.

Assume that there are s and t such that $q(x)s$ and $c(x)t$ are integers. Let $d = \max c(x) - \min c(x)$. Then for each β, there is

$$\alpha_1 \leq \beta \leq \alpha_2 \quad \textit{with} \quad \alpha_2 - \alpha_1 \geq \frac{1}{std^2}$$

such that $q(f(\alpha)) = q(f(\beta))$ and $q(f(\alpha)) = c(f(\beta))$ whenever $\alpha_1 \leq \alpha \leq \alpha_2$.

PROOF. Let us write $q(\alpha) = q(f(\alpha))$ and $c(\alpha) = c(f(\alpha))$. Define $g(\alpha) = q(\alpha) - \alpha c(\alpha)$.

Dinkelbach [1967] shows that $f(\alpha)$ and $c(\alpha)$ both decrease, proving the first part of the lemma. Moreover, the fact that $q(f(\alpha_1)) = q(f(\alpha_2))$ if and only if $c(f(\alpha_1)) = c(f(\alpha_2))$ follows immediately from the maximality of f.

Let $[\alpha_1, \alpha_2]$ be the maximal interval inside which $f(\alpha)$ and $c(\alpha)$ are constant.

Let $x = f(\beta)$. As shown by Dinkelbach [1967], g is continuous function. Since α_2 is maximal, there must be $y \in X$ such that $f(y) < f(x)$ and

$$f(x) - \alpha_2 g(x) = f(y) - \alpha_2 g(y).$$

Similarly, there is $z \in X$ with $f(z) > f(x)$ and

$$f(x) - \alpha_1 g(x) = f(z) - \alpha_1 g(z).$$

Let us write $u_1 = f(y) - f(x)$ and $u_2 = f(x) - f(z)$, and similarly $v_1 = g(y) - g(x)$ and $v_2 = g(x) - g(z)$. Then rewriting previous equalities leads to

$$\alpha_2 = \frac{f(y) - f(x)}{g(y) - g(x)} = \frac{u_1}{v_1} \quad \text{and} \quad \alpha_1 = \frac{f(x) - f(z)}{g(x) - g(z)} = \frac{u_2}{v_2}.$$

Combining the equalities leads to

$$\begin{aligned} \alpha_2 - \alpha_1 &= \frac{u_1 v_2 - v_1 u_2}{v_1 v_2} \\ &= \frac{1}{st} \frac{stu_1 v_2 - stv_1 u_2}{v_1 v_2} \\ &\geq \frac{1}{st} \frac{1}{v_1 v_2} \\ &\geq \frac{1}{std^2} \end{aligned}$$

which proves the lemma. □

PROOF OF PROPOSITION 4.4. Lemma A.1 immediately guarantees that there is an interval I of length ϵ around α^*, such that for any $\alpha \in I$, the corresponding solution yields the same quality as T^*. Since $\operatorname{span}(\mathcal{T}_l) > B$, we must have $\alpha_u \in I$. □

In order to prove Propositions 4.5 and 4.8 we need the following technical lemmas.

LEMMA A.2. *Assume a universe Ω and that we are given a sequence of pairs $X = (\omega_i, t_i), \ldots, (\omega_k, t_k)$, where $\omega_i \in \Omega$ and $t_i \in \mathbb{R}$. The function f defined as*

$$f(a, b) = |\{\omega_i \mid a \leq t_i \leq b\}|$$

is concave.

PROOF. Consider $a \leq b \leq c \leq d$, and define $X = \{\omega_i \mid a \leq t_i \leq c\}$ and $Y = \{\omega_i \mid b \leq t_i \leq d\}$. It is easy to check that

$$\begin{aligned} f(a,c) &= |X|, \\ f(b,d) &= |Y|, \\ f(a,d) &= |X \cup Y|, \text{ and} \\ f(b,c) &\leq |X \cap Y|. \end{aligned}$$

It follows that

$$\begin{aligned} f(a,d) + f(b,c) &\leq |X \cup Y| + |X \cap Y| \\ &= |X| + |Y| \\ &= f(a,c) + f(b,d), \end{aligned}$$

which proves the lemma. □

LEMMA A.3. *Assume function f having the form $f(a,b) = g(b-a)$, where g is concave function. Then f is concave.*

PROOF. Assume $a \leq b \leq c \leq d$, and write $z = b - a$, $x = c - b$, and $y = d - b$. Then $f(a,d) - f(b,d) = g(z+y) - g(y)$ and $f(a,c) - f(b,c) = g(z+x) - g(x)$. The concavity of g and the fact that $x \leq y$ now implies that $g(z+y) - g(y) \leq g(z+x) - g(x)$, which proves the lemma. □

PROOF OF PROPOSITION 4.5. Note that the gain is equal to

$$\begin{aligned} \delta_\alpha(T) &= q(W, \mathcal{T} \cup T, G) - \alpha \cdot \mathrm{span}(\mathcal{T} \cup T) \\ &= q(W, \mathcal{T} \cup T, G) - \alpha \cdot \mathrm{span}(\mathcal{T}) - \alpha \cdot \mathrm{span}(T)\,. \end{aligned}$$

The first term is concave due to Lemma A.2. The second term is constant. The third term is concave due to Lemma A.3. The sum of concave functions is concave, proving the proposition. □

PROOF OF PROPOSITION 4.8. Recall that $h(T,\beta) = g(T) - \beta \cdot c(T)$. The first term is concave due to Lemma A.2. The second term is concave due to Lemma A.3. The sum of concave functions is concave, proving the proposition. □

PROOF OF PROPOSITION 4.9. Let us first prove the correctness. Let T^* be the optimal time interval and let $\beta^* = g(T^*)/c(T^*)$.

Initial β_u is selected such that $h(T_u, \beta_u) \leq 0$. Moreover, initial β_l satisfies $h(T_l, \beta_l) \geq h(T_u, \beta_l) = 0$. It is easy to see that this invariant holds during the while-loop.

The while-loop can stop due to three different conditions. Case $h(T_l, \beta_l) = 0$. Here T_l is the optimal solution and will be returned by the algorithm. Case $g(T_l) = g(T_u)$. Lemma A.1 states that $g(T_l) \geq g(T^*) \geq g(T_u)$, which immediately implies $g(T_l) = g(T^*)$. Lemma A.1 guarantees that also $c(T_l) = c(T^*)$ which proves the correctness. Case $\beta_u < \beta_l + \epsilon$. Note that $g(T)w$ is an integer and $c(T)K'B'\delta$ is an integer for any T. Lemma A.1 implies that there is an interval I around β^* of length ϵ such that solving $h(T,\beta)$ for $\beta \in I$ will yield optimal result. We must have $\beta_u \in I$ or $\beta_l \in I$. This guarantees the correctness.

Let us now prove the time complexity. Since the difference $\beta_u - \beta_l$ is halved every iteration, this gives us first bound of $\mathcal{O}(\log wK'B'\delta)$.

To prove the other bound, first note that if $h(T_l, \beta_l) > 0$, then $g(T_l)/c(T_l) > \beta_l$. Since new β_l is always larger or equal than $g(T_l)/c(T_l)$, then β_l and $g(T_l)/c(T_l)$ increase during each iteration. Lemma A.1 implies that $g(T_l)$ decreases every iteration. There are at most p possible values for g. Consequently, the while-loop requires at most $\mathcal{O}(p)$ iterations. □