



FastMinTC+: A Fast and Effective Heuristic for Minimum Timeline Cover on Temporal Networks

Giorgio Lazzarinetti  

Università degli Studi Milano-Bicocca, Milano, Italy

Sara Manzoni  

Università degli Studi Milano-Bicocca, Milano, Italy

Italo Zoppis  

Università degli Studi Milano-Bicocca, Milano, Italy

Riccardo Dondi  

Università degli Studi di Bergamo, Bergamo, Italy

Abstract

The analysis and summarization of temporal networks are crucial for understanding complex interactions over time, yet pose significant computational challenges. This paper introduces FASTMINTC+, an innovative heuristic approach designed to efficiently solve the Minimum Timeline Cover (MINTCOVER) problem in temporal networks. Our approach focuses on the optimization of activity timelines within temporal networks, aiming to provide both effective and computationally feasible solutions. By employing a low-complexity approach, FASTMINTC+ adeptly handles massive temporal graphs, improving upon existing methods. Indeed, comparative evaluations on both synthetic and real-world datasets demonstrate that our algorithm outperforms established benchmarks with remarkable efficiency and accuracy. The results highlight the potential of heuristic approaches in the domain of temporal network analysis and open up new avenues for further research incorporating other computational techniques, for example deep learning, to enhance the adaptability and precision of such heuristics.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Design and analysis of algorithms; Theory of computation → Mathematical optimization; Theory of computation → Discrete optimization

Keywords and phrases Temporal Networks, Activity Timeline, Timeline Cover, Vertex Cover, Optimization, Heuristic

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.20

1 Introduction

Modern applications are increasingly incorporating new data abstractions that necessitate redefined approaches to data summarization and synthesis. Notably, with the widespread availability of temporal information, many datasets, traditionally modeled as networks, are now being treated as temporal networks [10, 18], i.e., graphs $G = (V, E)$ that include temporal edges representing interactions among a set of entities V , where each edge $(u, v, t) \in E$ captures the interaction at time t between entities u and v .

This paper introduces a novel heuristic, FastMinTC+, aimed at summarizing temporal networks – a critical area for data compression, visualization, interactive analysis, and noise reduction. Temporal network summarization poses unique challenges stemming from their inherent complexity and the diverse objectives of summarization, for which different methods have been proposed [17]. These methods employ a variety of techniques, including temporal motifs [19], graphlets [11], vocabulary-based summaries [26], evolutionary patterns [27], and community evolution [21]. While effective, such techniques can be complex and difficult to interpret. To simplify, research has shifted towards using activity time intervals to represent



© Giorgio Lazzarinetti, Sara Manzoni, Italo Zoppis, and Riccardo Dondi;
licensed under Creative Commons License CC-BY 4.0

31st International Symposium on Temporal Representation and Reasoning (TIME 2024).

Editors: Pietro Sala, Michael Sioutis, and Fusheng Wang; Article No. 20; pp. 20:1–20:18

Leibniz International Proceedings in Informatics

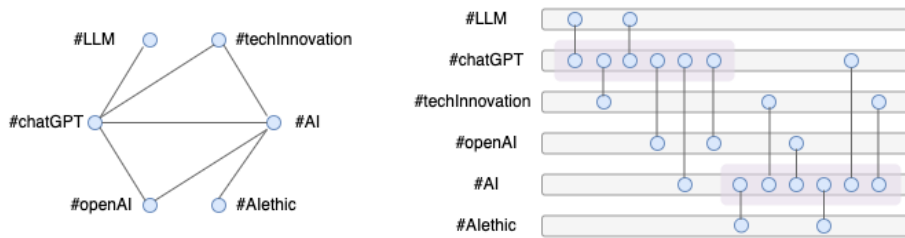


LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

20:2 FastMinTC+: An Heuristic for the MinTCover Problem

interactions between entities [24, 25, 8, 5, 4, 6, 7]. An interaction between two entities is accounted for if, at the time of their interaction, at least one of the entities is active. This summarization task involves identifying these latent activity intervals for all entities, producing an activity timeline that encompasses the entire network. This problem, known as *Minimum Timeline Cover* (MINTCOVER) problem, was introduced by Rozenstein et al. [25], with the goal of identifying crucial time intervals that elucidate significant network events.

To illustrate the importance of activity timelines in understanding events, consider the launch of *ChatGPT* by *OpenAI* in November 2022. This event, which quickly captured public attention due to its advances in *AI* and large language model (*LLM*) capabilities, sparked widespread social media engagement and discussions on *AI ethics* and potential *technological innovations*. Figure 1 shows a co-occurrence graph on the left, where vertices represent hashtags and edges connect hashtags that appear together in posts. On the right side of Figure 1, a temporal network model visualizes these interactions, highlighting how data structured in timelines of (entity, time-interval) pairs can offer deep insights into significant events. These timelines, indicated in purple, outline key moments such as the initial launch and subsequent debates around *AI*, underlining the central entities' roles in shaping discussions. This method of mapping event timelines is central to solving the MINTCOVER problem.



■ **Figure 1** Example of a co-occurrence graph (left side) and temporal network framework (right side) with event timelines (in purple). The analysis of hashtags coming from social media discussions on two distinct events in time intuitively shows the relationship between activity timelines and events, which are the objective of the MINTCOVER problem.

Despite the interpretability and effectiveness of this problem, studies on the hardness and parametrized complexity highlight that MINTCOVER is NP-hard and, when considering more than one time interval, not even approximable within any constant factor (deciding whether there exists a solution of span 0 is indeed an NP-complete problem) [25, 8, 5, 4]. From hence, our research focuses on the development of approximation and heuristic algorithms capable of generating satisfactory timeline covers in feasible time frames. In this study, we introduce a novel local search heuristic for the MINTCOVER problem, employing a low-complexity approach in order to make it feasible even on large-scale graphs. The performance of our method are shown with an experimental evaluation on both synthetic and real-world datasets. The experiments show that our algorithm improves upon existing methods, both for efficiency and accuracy.

The rest of the paper is organized as follow. In Section 2 we formally define the MINTCOVER problem and we present some related works on approximate solutions. In Section 3, we describe our heuristic, FASTMINTC+, for solving the MINTCOVER problem. In Section 4 we provide the experimental results on the outlined comparison. Final considerations and future direction are described in Section 5.

2 Preliminaries

2.1 Problem Definition and Notions

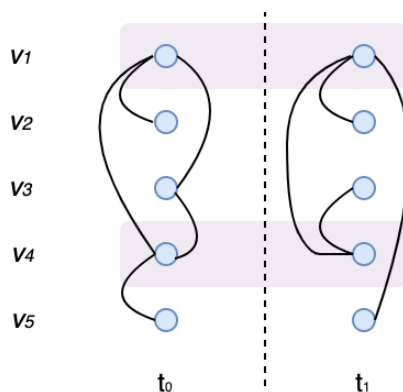
Let $G = (V, E)$ be a temporal graph, with V a set of vertices and E a set of temporal edges, where each edge is a triple $(u, v, t) \in E$, such that $(u, v) \in V$ and t is a timestamp indicating the time that an interaction between vertices u and v takes place. We consider unweighted undirected graphs. Given a vertex $u \in V$ we define, $E(u) = \{(u, v, t) \in E\}$ as the set of temporal edges incident in u , $N((u, t)) = \{v | (u, v, t) \in E\}$ as the set of vertices incident in u at timestamp t , $T(u) = \{t | (u, v, t) \in E\}$ as the set of timestamps of edges incident in u .

Following the definition provided in [7], given a temporal graph $G = (V, E)$ and a vertex $u \in V$, the *local degree* of u in a timestamp t , denoted as $deg_L((u, t)) = |N((u, t))|$, is the number of temporal edges incident in u at a timestamp t , while the *global degree* of a vertex u , denoted as $deg(u) = \sum_{t=1}^T deg_L((u, t))$ is the number of temporal edges incident in u in the overall time domain. Moreover, we define the overall *density* of an undirected graph as the ratio of the number of edges $|E|$ with respect to the maximum possible edges: $d = \frac{2|E|}{|V|(|V|-1)|T|}$. We consider graph to be *sparse* when $|E| = O(|V| + |T|)$.

Given two numbers s_u, e_u , with $s_u \leq e_u$ we define $I_u = [s_u, e_u]$ as the activity interval of vertex u and $\mathcal{T} = \{I_u\}_{u \in V}$ as an active timeline of G . Given an interval $I_u = [s_u, e_u]$, $\delta(I_u) = e_u - s_u$ is the *span* of interval I_u .

► **Definition 1** (Timeline Cover). *Given a temporal graph $G = (V, E)$ and an activity timeline $\mathcal{T} = \{I_u\}_{u \in V}$, we say that \mathcal{T} covers G if $\forall (u, v, t) \in E$, $t \in I_u$ or $t \in I_v$.*

Figure 2 shows an example of timeline covering over a 2-timestamps graph with 5 nodes.



■ **Figure 2** A temporal graph with 5 vertices and 9 edges, distributed over two timestamps. In purple we can see a possible timeline cover represented by intervals $I_{v1} = [0, 1]$ and $I_{v4} = [0, 1]$. This is a timeline cover since every edge $(u, v, t) \in E$ is such that t is in either I_u or in I_v .

The trivial timeline $I_u = [\min T(u), \max T(u)]$ provides a cover but may have unnecessarily long intervals. Indeed, the task is to find a timeline that has the most compact intervals possible according to some objective functions: the sum-span of a timeline \mathcal{T} , $S(\mathcal{T}) = \sum_{u \in V} \delta(I_u)$, or the max-span of a timeline, $\Delta\mathcal{T} = \max_{u \in V} \delta(I_u)$, are the objective functions proposed in the literature [25].

According to these quality measures, it is possible to define two problems:

► **Definition 2** (MINTCOVER₊). *Given a temporal network $G = (V, E)$, find a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ that covers G and minimizes the sum-span $S(\mathcal{T})$.*

► **Definition 3** (MINTCOVER_∞). *Given a temporal network $G = (V, E)$, find a timeline $\mathcal{T} = \{I_u\}_{u \in V}$ that covers G and maximize the max-span $\Delta\mathcal{T}$.*

The selection of either the SUM or MAX formulation of the problem is contingent upon the specific application context. Generally, the MAX formulation facilitates the derivation of a worst-case bound on the duration of all activity intervals; however, this approach is susceptible to outliers, whereby a single long interval may precipitate solutions characterized by disproportionately high costs. Conversely, the SUM formulation is advisable in scenarios characterized by considerable variability in the duration of events anticipated within the activity timeline.

These problems can be further extended to allow k active intervals per vertex. It has been shown that also when there is only one activity interval per vertex, i.e., $k = 1$, while the MAX problem can be reduced to 2-SAT, and solved optimally in linear time, the SUM problem is NP-hard [8]. For this reason, in this research, we focus on designing and implementing an heuristic algorithm for the SUM problem, with $k = 1$, given the scarcity of heuristics even for this case, with the goal of proposing an efficient and effective approach also for massive graphs. For this problem we provide an exact Integer Linear Programming (ILP) formulation in Appendix A, considered mainly to evaluate the performance of our method on small datasets. We will refer to this problem (MINTCOVER_+) as MINTCOVER .

2.2 Related Works

The MINTCOVER problem, being a recent and NP-hard problem, has limited researches which focus primarily on the study of the parametric complexity and the development of approximate algorithms aimed at theoretical outcomes. Thus, in the following we describe these approximate solutions and, since in building our heuristic we mainly based on established Minimum Vertex Cover (MVC) heuristics, we also introduce the state-of-the-art solutions for MVC.

2.2.1 Approximate Solutions

For MINTCOVER , few approximate solutions have been proposed. In [25], Rozenstein et al. propose an inner point iterative method as a strategy to solve this problem by initially considering a subproblem named COALESCE. This subproblem involves finding optimal activity timelines that include predetermined time points for each vertex, called *inner points*. These points are essentially estimates on where a vertex is presumed to be active. The challenge is to construct intervals around these points to cover all interactions. Remarkably, the authors develop a method to find a 2-approximate solution to the COALESCE problem in linear time, by first providing an ILP formulation, then by relaxing the integrality constraint to write the dual, whose solution is then used iteratively to refine the MINTCOVER algorithm. Each iteration adjusts the inner points based on the activity intervals derived from the COALESCE solution, until the changes in the solution become negligible, indicating that the intervals are covering all interactions with minimal total duration. This method not only ensures comprehensive interaction coverage but also strives to minimize the overall activity time across the network. If from the one hand the authors were able to find a 2-approximate solution to the COALESCE problem, they cannot provide an approximation factor for MINTCOVER , even though they can compute a solution for this latter problem in linear time.

An $O(T \log n)$ factor approximation algorithm, that consists of two main phases, is proposed by Dondi et al. in [6, 7]. The algorithm works by considering the union graph $G_u = (V, E_u)$ of the temporal graph $G = (V, E)$, which is a static labeled graph, where labels for each edge are the union of all the timestamps t associated to edges (u, v, t) , for each pair (u, v) . The first phase consists in finding a minimum vertex cover with a 2-factor approximation algorithm [12] for the subgraph of the union graph that contains edges with at least three labels. For each vertex of this set is defined an activity interval that spans the entire temporal graph starting from 1. The vertices in this set are then removed from the temporal graph G , resulting in a temporal graph G' such that each pair of vertices is connected by at most two temporal edges. Then the second phase, inspired by an approximation algorithm for SETCOVER [9], uses a randomized rounding algorithm to find an approximation solution to a variant of the problem called Minimum Non-Consecutive Timeline Cover (MIN-NC-TCOVER) where each vertex can be active in non consecutive timestamps. Then they define a solution of MINTCOVER where each vertex is active in an interval that includes the minimum and maximum timestamp where the vertex is active in the computed solution of MIN-NC-TCOVER.

2.2.2 Heuristics for Minimum Vertex Cover

MINTCOVER is a variant in temporal graphs of MVC, a classical NP-hard optimization problem that consists of, given an undirected unweighted graph $G = (V, E)$, finding a minimum sized subset $S \subseteq V$ such that every edge in G has at least one endpoint in S .

The MVC problem is often addressed by iteratively solving its decision version, which involves identifying a vertex cover of a specified size k . The process begins with the construction of a vertex cover. If a vertex cover of k vertices is found, one vertex is removed, reducing the target size to $k - 1$, and a local search is initiated to find a smaller vertex cover. The current candidate solution, denoted as C , includes the vertices selected for covering. Each vertex $v \in C$ has an associated *loss*, defined as the number of covered edges that would become uncovered if v were removed from C . Conversely, for vertices not in C , a *gain* is calculated based on the number of uncovered edges that would become covered upon their addition. Both loss and gain are used to score vertices, which also have an *age* indicating the time since their status last changed. The iterative process involves swapping vertices in C with those outside of it, a step known as *exchanging step*.

The FASTVC [3] (which in turn is inspired by NUMVC [2]) is a heuristic algorithm, known for its efficiency in handling large graphs, which makes it of practical relevance for real-world applications, where rapid solutions are needed. The algorithm works by following the procedure depicted before, adopting a two-stage exchange method as exchanging step, introduced in the NUMVC algorithm, which consists in firstly removing a vertex from C , then adding a new vertex not in C , updating the scoring properties (loss and gain) at each stage. The great advantage of FASTVC is that to enhance the efficiency and effectiveness of the vertex selection process it mainly relies on the Best from Multiple Selection (BMS) heuristic, which works by generating multiple candidate solutions, choosing the best among them based on a predefined criterion (the one that leads to the smallest vertex cover).

This approach is particularly useful since in problem like MVC the solution space is large and complex, thus direct evaluation of all possible selections would be computationally expensive. By using BMS, FASTVC can more effectively explore the solution space, as it avoids getting trapped in local minima – a common problem in greedy and local search algorithms. It provides a way to balance exploration and exploitation by periodically allowing the algorithm to consider a range of potential moves (vertex additions or removals) rather than being confined to the immediate best move.

More recently, the use of Deep Learning (DL) approaches has been proposed for defining heuristics for NP-hard problems over graphs, like MVC, highlighting that representation learning based approaches are better in building solution to combinatorial optimization over graphs with respect to end-to-end approaches [20]. Some of the most effective research investigation directions are based on using Reinforcement Learning (RL) [13, 1], Graph Neural Networks (GNNs) [16, 14] or the Attention Mechanism [15] by firstly learning a representation of the graph and then by leveraging this representation with autoregressive machine learning-based procedure, local search or greedy search to build the final solution.

3 FastMinTC+

In this section we describe our FASTMINTC+ algorithm, which solves the MINTCOVER problem in an iterative way, following a local search procedure and a heuristic optimization similar to the one adopted by FASTVC [3].

3.1 Overall Algorithm

The FASTMINTC+ overall algorithm (outlined in Algorithm 1) is composed of an *initialization step*, to compute an initial minimal timeline, and an *exchange step*, to reduce the span of the initial computed timeline.

■ **Algorithm 1** FastMinTC+(G , cutoff).

Input: graph $G = (V, E, T)$, *cutoff* time
Output: A minimal timeline cover of G , $\mathcal{T}^* = \{I_u\}_{u \in V}$
 $\mathcal{T}, loss \leftarrow \text{InitializeTC}(G)$
 $gain((v, t)) \leftarrow 0$ for each $(v, t) \notin \mathcal{T}$
 $\mathcal{T}^* \leftarrow \mathcal{T}$
while $elapsed \leq cutoff$ **do**
 if \mathcal{T} covers all edges AND $S(\mathcal{T}) < S(\mathcal{T}^*)$ **then**
 $\mathcal{T}^* \leftarrow \mathcal{T}$
 end
 $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(v, t) : loss((v, t)) \leq loss((u, t)), \forall (v, t), (u, t) \in \mathcal{T}\}$
 $(v, t) \leftarrow \text{SelectRndVertex}(\mathcal{T}, k)$
 $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(v, t)\}$
 $e \leftarrow$ a random uncovered edge
 $(v, t) \leftarrow$ the endpoint of e with greater gain breaking ties in favor of the one not included in the solution for a larger number of iterations
 $\mathcal{T} \leftarrow \mathcal{T} \cup \{(v, t)\}$
 update loss of vertices $N(v)$ in \mathcal{T} and gain of vertices $N(v)$ not in \mathcal{T}
end

The initialization step is carried out by the *InitializeTC* procedure, which returns an initial timeline which is granted to be minimal, and computes the loss value for each pair $(v, t) \in \mathcal{T}$, which corresponds to the number of edges that would become uncovered, by removing (v, t) from the computed timeline. We remember that the timeline \mathcal{T} is a set of intervals $I_v = [s_v, e_v]$, one for each vertex $v \in V$. Thus, we can consider \mathcal{T} as an ordered list of pairs (v, t) such that, for each vertex $v \in V$, we have exactly two pairs: (v, s_v) , (v, e_v) , with $s_v \leq e_v$. For each vertex in the timeline, thus, the loss is computed only for timestamps

s_v and e_v and not for intermediate timestamps in the interval. After letting computing the initial timeline and the corresponding *loss*, the initialization step computes the *gain*, which is defined for each vertex $(v, t) \notin \mathcal{T}$ as the number of uncovered edges that would become covered by adding (v, t) to the timeline.

The solution is then refined in the exchange step, with a two-stage exchange method, by firstly removing from the timeline the pair (vertex, timestamp) with minimum *loss* from the activity timeline computed and then by randomly reducing the length of the intervals selecting a second pair (vertex, timestamps) to be removed from the activity timeline with the *SelectRndVertex* procedure. Then, the algorithm picks a random uncovered edge e , and chooses the endpoint with greater gain adding it into \mathcal{T} , breaking ties in favor of the endpoint that have not been included in the solution for a larger number of iterations. Note that along with removing or adding a vertex, the *loss* and *gain* values of the vertex and its neighbors are updated accordingly.

3.2 Initialization Algorithm

The initialization procedure is outlined in Algorithm 2 and consists of an *extending phase* and a *shrinking phase*.

■ **Algorithm 2** InitializeTC(G).

```

Input: graph  $G = (V, E, T)$ 
Output: timeline cover of G  $\mathcal{T} = \{I_u\}_{u \in V}$ ,  $loss((v, t))$  for each  $(v, t) \in \mathcal{T}$ 
for  $e \in E$  do
    if  $e$  is uncovered then
        add the timestamp  $t$  to the activity interval of the endpoint of  $e$  with higher
        degree in  $t$ 
    end
end
// initialize loss to 0 (considering only the minimum and maximum  $t$  of each interval
of  $v$  in  $\mathcal{T}$ )
 $loss((v, t)) \leftarrow 0$  for each  $(v, t) \in \mathcal{T}$ 
for  $e \in E$  do
    if only one endpoint of  $e$  belongs to  $\mathcal{T}$  then
        for the endpoint  $(v, t) \in \mathcal{T}$ ,  $loss((v, t)) ++$ 
    end
end
// remove redundant vertices
for  $(v, t) \in \mathcal{T}$  do
    if  $s_v < e_v$  then
        if  $loss((v, t)) = 0$  then
             $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(v, t)\}$ 
            update loss of vertices  $N(v)$  in  $\mathcal{T}$ 
        end
    end
end

```

During the extending phase, the procedure begins with an empty set \mathcal{T} , progressively augmented by evaluating and incorporating edges one at a time. If an edge e is found to be uncovered, the endpoint of e with higher degree is added to \mathcal{T} . It is straightforward that

at the end of this phase we obtain a timeline cover. The shrinking phase consists in first computing the loss value of pairs (v, t) in \mathcal{T} , only for the initial and final timestamps of each interval (thus, we do not compute the loss for the intermediate timestamps in the interval, i.e., $\forall I_u = [s_u, e_u] \in \mathcal{T}$, we compute the loss only for (u, s_u) and (u, e_u)). This phase possibly shrinks an interval length, by removing the initial or final timestamp, if the loss is equal to 0. When computing the *loss* of (v, t) with respect to an edge (u, v, t) , we consider the entire interval on vertex u , i.e., $I_u = [s_u, e_u]$. If it holds that $s_u < t < e_u$, we do not increment the *loss* value of v since the edge (u, v, t) is already covered.

After computing the loss, the algorithm checks if there are vertices in the timeline with *loss* = 0. If it finds a (v, t) with *loss* = 0 and $I_v = [s_v, e_v]$ is such that $s_v < e_v$, then I_v is shrunked with a new timestamp r as follows: if $t = s_v$, s_v is replaced by r , which is the first timestamp in $T(v)$ greater than s_v (which may not necessary be $s_v + 1$ if v is not active in $s_v + 1$); if $t = e_v$, e_v is replaced by r , which is the next timestamp in $T(v)$ smaller than e_v (which may not necessary be $e_v - 1$ if v is not active in $e_v - 1$).

Subsequently *loss* of vertices in $N((v, t))$ is increased by 1 and *loss* of vertex (v, r) is computed considering the vertex in $N((v, r))$.

This algorithm grants to return a *minimal* solution according to Theorem 4 (proof is provided in Appendix B). A timeline cover is minimal if removing any (v, t) would make it not a timeline cover, thus the *loss* $(v, t) > 0$, $\forall (v, t) \in \mathcal{T}$.

► **Theorem 4.** *The timeline $\mathcal{T} = \{I_u\}_{u \in V}$ returned by the *initializeTC* algorithm is a minimal timeline cover.*

3.3 Random Vertex Selection Algorithm

A critical function for FASTMINTC+ is *SelectRndVertex*, outlined in Algorithm 3, which chooses a vertex from the candidate vertex set \mathcal{T} to remove.

■ **Algorithm 3** *SelectRndVertex*(\mathcal{T} , k).

Input: A timeline \mathcal{T} , a parameter k
Output: an element of \mathcal{T}
 $best \leftarrow$ a random vertex (v, t) from \mathcal{T}
for 1 to $k-1$ **do**
 $tmp \leftarrow$ a random vertex (u, r) from \mathcal{T}
 if $loss(tmp) < loss(best)$ **then**
 $best \leftarrow tmp$
 end
end
return $best$

This algorithm follows the structure of the cost-effective BMS heuristic, which picks k elements (where k is a parameter) randomly with replacement from the set \mathcal{T} , and then returns the one with lowest *loss* value. The set \mathcal{T} is composed of all the pairs (v, t) that belong to the activity timeline \mathcal{T} and whose timestamps are an initial or ending timestamp of an interval. In this way, we only remove from the timeline vertices which are at the beginning or end of the interval, with the goal of reducing the length of the activity timelines avoiding producing multiple intervals for the same vertex. From hence, Theorem 5 holds (proof is provided in Appendix C).

► **Theorem 5.** *With $k \geq 50$, the probability that the *SelectRndVertex* algorithm choose a vertex whose loss value is not greater than 90% vertices in \mathcal{T} is greater or equal to 0.9948.*

3.4 Heuristic Complexity Analysis

The overall FASTMINTC+ heuristic (outlined in Algorithm 1), as seen, is composed of two main parts: the initialization step (i.e., the *InitializeTC* procedure outlined in Algorithm 2, followed by the *gain* initialization) and the exchange step (i.e., the iterative cycle carried out for an *elapsed* time smaller or equal to a given *cutoff* time, that uses the *SelectRndVertex* procedure, outlined in Algorithm 3 as two-stage exchange method). Thus, the complexity of the FASTMINTC+ heuristic relies on the complexity of Algorithms 2 and 3.

In the following we denote $n = |V|$, $m = |E|$ and $t = |\mathcal{T}|$.

Let us first focus on the complexity of the *InitializeTC* algorithm. This can be divided in three parts: the extending phase, the initialization of the *loss* values and the shrinking phase. It is straightforward that the complexity of the extending phase is $O(m)$. Indeed, to compute the extending phase, it is necessary to scan the set of edges E one time, adding, for every scanned edge, the endpoint with higher degree. As far as *loss* computation is concerned, it depends on the number of vertices $(v, t) \in \mathcal{T}$. Since we add, for each vertex $v \in V$ exactly two timestamps (the start and end of the activity interval, namely, s_v, e_v), the dimension of the computed activity timeline is exactly $|\mathcal{T}| = 2n$. It follows that the complexity for the initialization of the loss is $O(n)$. In order to update the *loss* values, we must check if an edge is covered by one of two vertices of the timeline; we need to scan the set of edges E , the complexity is $O(m)$. Thus, the overall complexity of *loss* initialization is $O(n + m)$. For the shrinking phase, the complexity depends on the number of updates performed over the *loss* values. Since each vertex is updated at most once for each edge incident in it, the total number of possible updates is bounded to the sum of the global degrees of each vertex, i.e., $\sum_{v \in V} \text{deg}(v) = 2m$, thus the complexity of the shrinking phase is $O(m + n)$. Therefore, the overall complexity of the *InitializeTC* algorithm is $O(m + n)$.

Let's now move to the *SelectRndVertex* procedure. Since the algorithm only performs comparison operation for pair of elements at a time for each iteration, the complexity is $O(k)$, where k is the number of iteration. Since k is constant, the overall complexity is $O(1)$.

Consider now the FASTMINTC+ heuristic. From the previous analysis, it holds that the computational complexity of the initialization phase, i.e., the one derived from the *InitializeTC* algorithm, is $O(m + n)$. For the exchange step, the complexity can be derived by: 1) checking whether \mathcal{T} covers all edges, which can be done summing the *gain* values of $(v, t) \notin \mathcal{T}$ with complexity $O(n)$: if the sum is 0, it means that \mathcal{T} is a minimal cover; 2) getting the vertex (v, t) with lower *loss* value, which can be done with complexity $O(n)$; 3) select a random vertex with the *SelectRndVertex* procedure, which has complexity $O(1)$; 4) extract a random uncovered edge e and add the endpoint of e with greater *gain*, breaking ties in favor of the older one, which has the same complexity as the *loss* update operation, thus $O(m + n)$; 5) update *loss* and *gain* values, which have both complexity $O(m)$. Thus, the overall time complexity of the exchange step is $O(m + n)$, leading to an overall time complexity of the FASTMINTC+ heuristic equals to $O(m + n)$.

4 Performance Evaluation

In this section we provide the results of our experiments on FASTMINTC+. We evaluate it against state-of-the-art approaches on both synthetic and real world graphs.

4.1 Dataset Description

In the current literature, there are no publicly available datasets for the MINTCOVER problem. The sole study addressing experimental analysis on real and synthetic instances is presented in [25]. However, this study has significant limitations: firstly, the method proposed for generating synthetic datasets does not ensure that the computed ground truth is optimal, complicating the assessment of the algorithm’s performance against exact solvers; secondly, the real-world dataset employed is not publicly accessible, precluding reproducibility of the results. To address these gaps, we have developed three distinct datasets, each designed for specific evaluative purposes:

- *Dataset1* consists of 264 synthetically generated instances of sparse temporal graphs, characterized by a low temporal edge-to-vertex ratio ($|E| = O(|V| + |T|)$). These graphs vary considerably, with vertex counts $|V| = [10, 10000]$, timestamp sets $T = [4, 5000]$, and edges $|E| = [10, 1000000]$. This dataset facilitates comparisons between the proposed heuristic and state-of-the-art algorithms across sparse graph scenarios, which reflect conditions found in many real-world datasets. For more granular analysis, instances are categorized as *small* (up to 50 vertices and 20 timestamps), *medium* (up to 500 vertices and 500 timestamps), and *hard* (up to 10000 vertices and 5000 timestamps).
- *Dataset2* includes 195 synthetically generated instances of dense temporal graphs ($|E| \gg O(|V| + |T|)$). These instances also range in size with vertices $|V| = [10, 10000]$, timestamps $T = [2, 5000]$, and edges $|E| = [100, 20000000]$. The purpose of *Dataset2* mirrors that of *Dataset1*, but focuses on denser graphs. Similar to *Dataset1*, it is divided into *small* (up to 30 vertices and 4 timestamps), *medium* (up to 1000 vertices and 100 timestamps), and *hard* categories (up to 10000 vertices and 5000 timestamps) to account for scalability concerns.
- *Dataset3* comprises 25 publicly available instances of temporal benchmark graphs sourced from the DIMACS repository [22]. This dataset was curated to include a diverse array of graphs in terms of edge count $|E|$, vertex count $|V|$ and timestamp range $|T|$, thus resulting in a set of graphs with densities D that ranges from $8,49 \text{ E-}10$ to $4,74 \text{ E-}01$. The primary aim of *Dataset3* is to evaluate the performance of the proposed heuristic against state-of-the-art algorithms on benchmark graphs, thereby assessing their applicability to practical scenarios.

4.2 Experimental Results

To evaluate our heuristic FASTMINTC+ we benchmark it against both an optimal solution derived from the ILP formulation (Equation 1 in Appendix A) and the principal approximation algorithms discussed in Section 2.2.1. The compared algorithms include the iterative method for inner points (INNER) as outlined in [25], and the two-phases approach (2PHASES) introduced in [7]. Our evaluations span three datasets, assessing both the quality (measured by the length of the sum-span of the timelines $S(\mathcal{T})$) and scalability (determined by execution time) of each solution. The algorithmic comparison varies by dataset and instance complexity; for *Dataset1* and *Dataset2*, all four algorithms are evaluated on small instances. However, the ILP solution is omitted from medium instances and both the ILP and 2PHASES are excluded from hard instances due to computational limitations. Only INNER and FASTMINTC+ are analyzed for *Dataset3* due to their superior performance.

In running the test, for the FASTMINTC+ algorithm, we set the parameter $k = 50$ and instead of setting a time limit, we define a number of 2000 iteration for the exchange step. The algorithm is executed 5 times each with a different shuffling of the edges, and the best result is reported. For the INNER algorithm we set the number of iterations to 10 (as suggested by the authors). No parameters have to be set for the 2PHASES approximation.

We implement our algorithm FASTMINTC+ in the Python programming language (version 3.12). The code for INNER is open-source and implemented in Python (version 2) [23]. For the 2PHASES algorithm no implementation were available online, thus, we implement it in Python (version 3.12). Experiments are carried out on a MacBook Pro (2017) under MacOS, using a 16GB RAM and 4 cores of a i7-3,1 GHz CPU.

4.2.1 Sparse Instances Results

The comprehensive results for *Dataset1*, consisting of sparse graphs, are summarized in Table 1. FASTMINTC+ consistently outperforms both INNER and 2PHASES in small instances. Specifically, compared to the sum-span calculated by the ILP and other algorithms, FASTMINTC+ approximates the ILP-derived solution more closely than its competitors. Among the 189 simple instances in *Dataset1*, FASTMINTC+ provides a superior solution (i.e., a shorter sum-span) compared to INNER 177 times and to 2PHASES 154 times. In the subset of 25 medium instances, FASTMINTC+ continues to outperform, invariably offering better solutions than 2PHASES and surpassing INNER in 14 instances. Similar trends are observed in the 50 hard instances, where FASTMINTC+ excels over INNER 34 times.

The fact that 2PHASES’s performance downgrade with larger instances, aligns with the fact that it provides an approximate factor proportional to the number of timestamps.

A notable finding from this experiment pertains to the average execution times. Clearly, the ILP algorithm, while providing optimal solutions, is significantly slower even for basic sparse instances. 2PHASES encounters scalability issues, notably in hard instances where experiments could not be conducted due to prohibitive execution times. Conversely, FASTMINTC+ is faster, even in large sparse instances, typically requiring an order of magnitude less time to compute the solution compared to INNER.

Thus, although FASTMINTC+ generally outperforms INNER and 2PHASES, it may not be invariably the superior choice for sparse graphs, unless graph size increases to a point where INNER becomes computationally impractical.

■ **Table 1** Experimental Results on Dataset1 - small, medium and hard instances. Results show that the proposed heuristic produces for sparse graphs better results with respect to the approximate solutions in a smaller time.

| Dataset1 | Algorithm | Average Execution Time | Average Sum Span |
|------------------|------------|------------------------|------------------|
| Small Instances | ILP | 1,13 E-01 | 1,01 E+00 |
| | 2Phases | 8,68 E-02 | 3,37 E+00 |
| | Inner | 9,37 E-04 | 7,05 E+00 |
| | FastMinTC+ | 5,39 E-04 | 3,02 E+00 |
| Medium Instances | 2Phases | 3,52 E+01 | 80,1 E+03 |
| | Inner | 1,21 E-01 | 71,1 E+03 |
| | FastMinTC+ | 4,32 E-02 | 70,9 E+03 |
| Hard Instances | Inner | 1,75 E+02 | 16,01 E+06 |
| | FastMinTC+ | 1,43 E+01 | 15,95 E+06 |

4.2.2 Dense Instances Results

The comprehensive results for *Dataset2*, composed of dense graphs, are summarized in Table 2. The findings from evaluations of small, medium, and hard instances are parallel to those observed in *Dataset1*, reinforcing the quality and the scalability of the proposed heuristic.

20:12 FastMinTC+: An Heuristic for the MinTCover Problem

Notably, the difference in average sum-span and execution time between FASTMINTC+ on both approximate and exact solutions is more pronounced here than in *Dataset1*. Indeed, as far as sum-span is concerned, in all 195 test cases assessed, INNER only outperformed FASTMINTC+ once in a small instance scenario, while as far as execution time is concerned, FASTMINTC+ always outperforms INNER, reaching, on average, a larger delta with respect to tests on *Dataset1*, with a maximum delta of 3492,17 seconds on the instance with 10000 vertices and 5000 timestamps. These results validate FASTMINTC+'s capability to effectively scale to larger instances. Moreover, they suggest a distinct advantage in favor of FASTMINTC+ when handling dense graphs, indicating its overall preferable performance relative to INNER in such contexts.

■ **Table 2** Experimental Results on Dataset2 - small, medium and hard instances. Results confirm that the proposed solution produces for dense graphs better results with respect to the approximate solutions in a smaller time.

| Dataset2 | Algorithm | Average Execution Time | Average Sum Span |
|------------------|------------|------------------------|------------------|
| Small Instances | ILP | 9,62 E+01 | 8,24 E+00 |
| | 2Phases | 3,29 E-01 | 27,85 E+00 |
| | Inner | 2,68 E-03 | 20,39 E+00 |
| | FastMinTC+ | 5,73 E-04 | 11,88 E+00 |
| Medium Instances | 2Phases | 8,54 E+00 | 28,69 E+03 |
| | Inner | 5,81 E-01 | 28,38 E+03 |
| | FastMinTC+ | 9,81 E-02 | 27,08 E+03 |
| Hard Instances | Inner | 8,96 E+02 | 14,74 E+06 |
| | FastMinTC+ | 1,82 E+01 | 14,05 E+06 |

4.2.3 Real World Instances Results

Experiments conducted on real-world graphs corroborate the findings observed in the synthetic test instances too. Detailed results for these real-world instances are presented in Tables 3 and 4. For each graph considered, Table 3 provides the results in terms of sum-span, while Table 4 provides the results in terms of execution time.

Overall, FASTMINTC+ demonstrates superior performance in terms of sum-span compared to INNER on non-sparse graphs. Specifically, in the analysis of 25 real-world instances, FASTMINTC+ achieves a better sum-span in 17 cases. Notably, the instances where INNER outperforms FASTMINTC+ are characterized by particularly low densities, with a maximum density value of 6,27 E-04 and an average density value of 1,09 E-04. In contrast, the instances where FASTMINTC+ outperforms INNER are characterized by a maximum density value of 4,47 E-01 and an average density value of 1,26 E-01. It is also worth noting that there are, albeit rarer, cases of graphs with very low densities (such as *ia-contacts-dublin* with $D = 8.98E - 08$) where FASTMINTC+ outperforms INNER, while there are no cases of dense graphs where INNER outperforms FASTMINTC+. This confirms that, while FASTMINTC+ should be always preferable in the case of dense graphs, it should be taken into account also in the case of sparse graphs, representing a valid alternative to INNER even in these cases.

The results also confirm the best performance in terms of execution time of the FASTMINTC+, which always achieves a lower execution time compared to INNER.

■ **Table 3** Experimental Results on real-world publicly available graphs from Dataset3. For each considered graph we report the information on the size of the network and the results in terms of sum-span $S(\mathcal{T})$ of INNER and FASTMINTC+.

| Graph | $ E $ | $ V $ | $ T $ | D | Inner $S(\mathcal{T})$ | FMTC+ $S(\mathcal{T})$ |
|-------------------------------|----------|--------|--------|----------|---------------------------|---------------------------|
| aves-sparrow-social | 516 | 52 | 1 | 3,89E-01 | 1,40E+01 | 1,00E+01 |
| aves-wildbird-network | 11900 | 202 | 5 | 1,17E-01 | 5,88E+02 | 4,78E+02 |
| copresence-InVS13 | 394247 | 95 | 20128 | 4,39E-03 | 1,53E+06 | 1,49E+06 |
| copresence-InVS15 | 1283194 | 219 | 21535 | 2,50E-03 | 3,71E+06 | 3,64E+06 |
| copresence-LH10 | 150126 | 73 | 12604 | 4,53E-03 | 3,97E+05 | 3,87E+05 |
| copresence-LyonSchool | 6594492 | 242 | 3123 | 7,24E-02 | 7,02E+05 | 6,64E+05 |
| copresence-SFHH | 1417485 | 403 | 3148 | 5,56E-03 | 9,26E+05 | 8,92E+05 |
| copresence-Thiers13 | 18613039 | 328 | 8937 | 3,88E-02 | 2,57E+06 | 2,45E+06 |
| email-dnc | 39264 | 1892 | 19382 | 1,13E-06 | 2,54E+05 | 5,84E+05 |
| fb-wosn-friends | 1269502 | 63731 | 736674 | 8,49E-10 | 8,99E+08 | 9,83E+08 |
| ia-contacts-dublin | 415912 | 10972 | 76943 | 8,98E-08 | 1,00E+06 | 9,60E+05 |
| ia-contacts-hypertext2009 | 20818 | 113 | 5245 | 6,27E-04 | 3,50E+05 | 3,93E+05 |
| ia-digg-reply | 87627 | 30398 | 83942 | 2,26E-09 | 1,41E+08 | 1,76E+08 |
| ia-hospital-ward-proximity | 32424 | 75 | 9452 | 1,24E-03 | 3,26E+05 | 3,18E+05 |
| ia-primary-school-proximity | 125773 | 242 | 3099 | 1,39E-03 | 6,51E+05 | 6,30E+05 |
| a-prosper-loans | 3394979 | 89269 | 1258 | 6,77E-07 | 9,78E+05 | 1,00E+06 |
| a-retweet-pol | 61157 | 18470 | 60500 | 5,93E-09 | 1,06E+07 | 1,21E+07 |
| insecta-ant-colony1 | 111578 | 113 | 40 | 4,41E-01 | 3,46E+03 | 3,25E+03 |
| insecta-ant-colony3 | 241280 | 160 | 40 | 4,74E-01 | 5,07E+03 | 4,77E+03 |
| insecta-ant-colony5 | 194317 | 152 | 40 | 4,23E-01 | 4,36E+03 | 4,14E+03 |
| mammalia-raccoon-proximity | 1997 | 24 | 51 | 1,42E-01 | 7,73E+02 | 7,61E+02 |
| rec-amz-Baby | 915446 | 596316 | 4868 | 1,06E-09 | 2,80E+06 | 2,91E+06 |
| reptilia-tortoise-network-bsv | 554 | 136 | 3 | 2,01E-02 | 9,60E+01 | 5,00E+01 |
| reptilia-tortoise-network-fi | 1713 | 787 | 8 | 6,92E-04 | 5,5E+02 | 3,12E+02 |
| SFHH-conf-sensor | 70261 | 403 | 3508 | 2,47E-04 | 7,11E+05 | 8,27E+05 |

■ **Table 4** Experimental Results on real-world publicly available graphs from Dataset3. For each considered graph we report the information on the size of the network and the results in terms of execution time of INNER and FASTMINTC+.

| Graph | E | V | T | D | Inner elapsed | FMTC+ elapsed |
|-------------------------------|----------|--------|--------|----------|---------------|---------------|
| aves-sparrow-social | 516 | 52 | 1 | 3,89E-01 | 0,01 | 0,00 |
| aves-wildbird-network | 11900 | 202 | 5 | 1,17E-01 | 0,09 | 0,05 |
| copresence-InVS13 | 394247 | 95 | 20128 | 4,39E-03 | 2,52 | 1,07 |
| copresence-InVS15 | 1283194 | 219 | 21535 | 2,50E-03 | 8,50 | 3,47 |
| copresence-LH10 | 150126 | 73 | 12604 | 4,53E-03 | 1,00 | 0,39 |
| copresence-LyonSchool | 6594492 | 242 | 3123 | 7,24E-02 | 43,65 | 15,92 |
| copresence-SFHH | 1417485 | 403 | 3148 | 5,56E-03 | 9,59 | 3,18 |
| copresence-Thiers13 | 18613039 | 328 | 8937 | 3,88E-02 | 114,17 | 43,69 |
| email-dnc | 39264 | 1892 | 19382 | 1,13E-06 | 0,39 | 0,13 |
| fb-wosn-friends | 1269502 | 63731 | 736674 | 8,49E-10 | 11,36 | 7,19 |
| ia-contacts-dublin | 415912 | 10972 | 76943 | 8,98E-08 | 3,38 | 1,12 |
| ia-contacts-hypertext2009 | 20818 | 113 | 5245 | 6,27E-04 | 0,13 | 0,06 |
| ia-digg-reply | 87627 | 30398 | 83942 | 2,26E-09 | 0,98 | 1,49 |
| ia-hospital-ward-proximity | 32424 | 75 | 9452 | 1,24E-03 | 0,21 | 0,09 |
| ia-primary-school-proximity | 125773 | 242 | 3099 | 1,39E-03 | 0,80 | 0,32 |
| a-prosper-loans | 3394979 | 89269 | 1258 | 6,77E-07 | 36,98 | 23,71 |
| a-retweet-pol | 61157 | 18470 | 60500 | 5,93E-09 | 0,73 | 0,77 |
| insecta-ant-colony1 | 111578 | 113 | 40 | 4,41E-01 | 0,66 | 0,31 |
| insecta-ant-colony3 | 241280 | 160 | 40 | 4,74E-01 | 1,39 | 0,61 |
| insecta-ant-colony5 | 194317 | 152 | 40 | 4,23E-01 | 1,12 | 0,50 |
| mammalia-raccoon-proximity | 1997 | 24 | 51 | 1,42E-01 | 0,01 | 0,01 |
| rec-amz-Baby | 915446 | 596316 | 4868 | 1,06E-09 | 18,16 | 12,53 |
| reptilia-tortoise-network-bsv | 554 | 136 | 3 | 2,01E-02 | 0,14 | 0,64 |
| reptilia-tortoise-network-fi | 1713 | 787 | 8 | 6,92E-04 | 0,02 | 0,01 |
| SFHH-conf-sensor | 70261 | 403 | 3508 | 2,47E-04 | 0,45 | 0,19 |

5 Conclusion and Future Works

This research tackled the complex problem of summarizing temporal networks, aiming to optimize the timeline cover for entities within a temporal graph. We introduced a novel heuristic approach, FASTMINTC+, which significantly advances the field by offering a computationally feasible solution to the MINTCOVER problem. Our method leverages low-complexity approximate heuristics, shown to be very effective on the related MVC problem on static networks, enabling the effective processing of massive graphs, a notable improvement over existing methodologies.

Experimental results, on both synthetic and real-world datasets, demonstrates that FASTMINTC+ achieves superior performance compared to state-of-the-art algorithms, enhancing computational efficiency and maintaining a high level of accuracy and reliability in identifying sub-optimal timeline covers. These results underscore the potential of our heuristic to facilitate deeper insights into temporal data analysis.

Looking ahead, the integration of DL techniques presents a promising avenue for further enhancing the efficacy of our heuristic. Specifically, exploring the synergy between our heuristic approaches and DL models could yield innovative strategies for tackling the MINTCOVER problem. These developments are poised to redefine the boundaries of temporal network analysis, opening up new possibilities for both theoretical and practical applications in the field.

References

- 1 Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL: <https://openreview.net/forum?id=Bk9mx1SFx>.
- 2 Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 747–753. AAAI Press, 2015. URL: <http://ijcai.org/Abstract/15/111>.
- 3 Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. Numvc: An efficient local search algorithm for minimum vertex cover. *J. Artif. Intell. Res.*, 46:687–716, 2013. doi:10.1613/JAIR.3907.
- 4 Riccardo Dondi. Untangling temporal graphs of bounded degree. *Theor. Comput. Sci.*, 969:114040, 2023. doi:10.1016/J.TCS.2023.114040.
- 5 Riccardo Dondi and Manuel Lafond. An FPT algorithm for temporal graph untangling. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPICs*, pages 12:1–12:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.IPEC.2023.12.
- 6 Riccardo Dondi and Alexandru Popa. Timeline cover in temporal graphs: Exact and approximation algorithms. In Sun-Yuan Hsieh, Ling-Ju Hung, and Chia-Wei Lee, editors, *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7-10, 2023, Proceedings*, volume 13889 of *Lecture Notes in Computer Science*, pages 173–184. Springer, 2023. doi:10.1007/978-3-031-34347-6_15.
- 7 Riccardo Dondi and Alexandru Popa. Exact and approximation algorithms for covering timeline in temporal graphs. *Annals of Operations Research*, April 2024. doi:10.1007/s10479-024-05993-8.

- 8 Vincent Froese, Pascal Kunz, and Philipp Zschoche. Disentangling the computational complexity of network untangling. *CoRR*, abs/2204.02668, 2022. doi:10.48550/arXiv.2204.02668.
- 9 Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.*, 11(3):555–556, 1982. doi:10.1137/0211045.
- 10 Petter Holme and Jari Saramäki. Temporal networks. *CoRR*, abs/1108.1780, 2011. arXiv:1108.1780.
- 11 Yuriy Hulovaty, Huili Chen, and Tijana Milenkovic. Exploring the structure and function of temporal networks with dynamic graphlets. *Bioinform.*, 32(15):2402, 2016. doi:10.1093/BIOINFORMATICS/BTW310.
- 12 George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Trans. Algorithms*, 5(4):41:1–41:8, 2009. doi:10.1145/1597036.1597045.
- 13 Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6348–6358, 2017. URL: <https://proceedings.neurips.cc/paper/2017/hash/d9896106ca98d3d05b8cbdf4fd8b13a1-Abstract.html>.
- 14 Kenneth Langedal, Johannes Langguth, Fredrik Manne, and Daniel Thilo Schroeder. Efficient minimum weight vertex cover heuristics using graph neural networks. In Christian Schulz and Bora Uçar, editors, *20th International Symposium on Experimental Algorithms, SEA 2022, July 25-27, 2022, Heidelberg, Germany*, volume 233 of *LIPICs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SEA.2022.12.
- 15 Giorgio Lazzarinetti, Riccardo Dondi, Sara Manzoni, and Italo Zoppis. An attention-based method for the minimum vertex cover problem on complex networks. *Algorithms*, 17(2):72, 2024. doi:10.3390/A17020072.
- 16 Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 537–546, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/8d3bba7425e7c98c50f52ca1b52d3735-Abstract.html>.
- 17 Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph summarization methods and applications: A survey. *ACM Comput. Surv.*, 51(3):62:1–62:34, 2018. doi:10.1145/3186727.
- 18 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.
- 19 Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. Motifs in temporal networks. *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2016. URL: <https://api.semanticscholar.org/CorpusID:13332080>.
- 20 Yun Peng, Byron Choi, and Jianliang Xu. Graph learning for combinatorial optimization: A survey of state-of-the-art. *Data Sci. Eng.*, 6(2):119–141, 2021. doi:10.1007/S41019-021-00155-3.
- 21 Anna-Kaisa Pietiläinen and Christophe Diot. Dissemination in opportunistic social networks: the role of temporal communities. In *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '12*, pages 165–174, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2248371.2248396.
- 22 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL: <https://networkrepository.com>.
- 23 P. Rozenshtein. the-network-untangling-problem. <https://github.com/polinapolina/the-network-untangling-problem/tree/master>, 2020. [Online; accessed 13-June-2024].

- 24 Polina Rozenshtein, Francesco Bonchi, Aristides Gionis, Mauro Sozio, and Nikolaj Tatti. Finding events in temporal networks: segmentation meets densest subgraph discovery. *Knowl. Inf. Syst.*, 62(4):1611–1639, 2020. doi:10.1007/S10115-019-01403-9.
- 25 Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. The network-untangling problem: from interactions to activity timelines. *Data Min. Knowl. Discov.*, 35(1):213–247, 2021. doi:10.1007/S10618-020-00717-5.
- 26 Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1055–1064. ACM, 2015. doi:10.1145/2783258.2783321.
- 27 Bianca Wackersreuther, Peter Wackersreuther, Annahita Oswald, Christian Böhm, and Karsten M. Borgwardt. Frequent subgraph discovery in dynamic networks. In Ulf Brefeld, Lise Getoor, and Sofus A. Macskassy, editors, *Proceedings of the Eighth Workshop on Mining and Learning with Graphs, MLG '10, Washington, D.C., USA, July 24-25, 2010*, pages 155–162. ACM, 2010. doi:10.1145/1830252.1830272.

A ILP formulation for MinTCover

The MINTCOVER problem can be formulated as an ILP mainly considering a variable $x_{u,t} \in \{0, 1\}$ whose value is 1 if t is included in the activity interval of u , 0 otherwise. With this variable we can formulate the following constraints:

- *Edge Coverage*: For each edge $(u, v, t) \in E$, at least one between u and v must have t in its activity interval. This can be expressed as $x_{u,t} + x_{v,t} \geq 1 \forall (u, v, t) \in E$.
- *Activity Interval definition*: for each vertex u and for each timestamp t , if t is included in the activity interval of u , then the following should hold: $s_u \leq t \leq e_u$. This means that if $x_{u,t} = 1$, then $s_u \leq t$ and $e_u \geq t$. This can be expressed, by adding a constant $M \geq \max(T)$, with two conditions, one over s_u and one over e_u as follows:

1. $s_u \leq t * x_{u,t} + M(1 - x_{u,t})$
2. $e_u \geq t * x_{u,t} - M(1 - x_{u,t})$

Indeed, if $x_{u,t} = 0$, it holds that $M \geq s_u$ and $-M \leq e_u$, otherwise it holds that $s_u \leq t$ and $e_u \geq t$.

- *Length of the Interval*: for each vertex u , the length of the activity interval is by definition $\delta(I_u) = e_u - s_u$. This is the objective function to be minimized. For simplicity, we can define $\delta(I_u) = d_u$.

With these variables and constraints we can formulate the ILP as in Equation 1.

$$\begin{aligned}
 & \text{minimize} && \sum_{u \in V} d_u \\
 & \text{subject to} && x_{u,t} + x_{v,t} \geq 1 && \forall (u, v, t) \in E \\
 & && s_u \leq t * x_{u,t} + M(1 - x_{u,t}) && \forall u \in V, \quad \forall t \in T \\
 & && e_u \geq t * x_{u,t} - M(1 - x_{u,t}) && \forall u \in V, \quad \forall t \in T
 \end{aligned} \tag{1}$$

B InitializeTC's minimality: Proof

Theorem 4 states that the timeline $\mathcal{T} = \{I_u\}_{u \in V}$ returned by the *initializeTC* algorithm, outlined in Algorithm 2, is a minimal timeline cover.

Consider the timeline $\mathcal{T} = \{I_u\}_{u \in V}$ produce as output by the extending phase, which may be modified in the shrinking phase. By construction, at the end of the extending phase \mathcal{T} is a timeline cover. We now prove Theorem 4 by showing that: 1) the timeline \mathcal{T} returned by the procedure is a timeline cover; 2) loss value of any vertex in the timeline \mathcal{T} does not decrease; 3) the timeline \mathcal{T} returned after the shrinking phase is minimal.

Proof.

- 1) Suppose to perform i^{th} iteration of the shrinking phase. Note that, if \mathcal{T} is a timeline cover at the i^{th} iteration, it is a timeline cover also at the $(i+1)^{\text{th}}$ iteration. Indeed, if $\text{loss}(v, t)_{i+1} > 0$, then \mathcal{T} does not change; if $\text{loss}(v, t)_{i+1} = 0$, then (v, t) is removed, but according to the definition of loss , removing such vertex (v, t) would not generate any new uncovered edge. From hence, since \mathcal{T} is a timeline cover at iteration 0, it is a timeline cover also the end of the shrinking phase (i^{th} iteration).
- 2) Notice that during the shrinking phase, the loss value of any vertex (v, t) in \mathcal{T} does not decrease. Indeed, if at the i^{th} iteration $\text{loss}(v, t)_i > 0$ the iteration does nothing, thus its loss value does not decrease; if $\text{loss}(v, t)_i = 0$, then all vertices in $N(v)$ belong to \mathcal{T} , otherwise, if at the i^{th} iteration there exists a vertex $(u, t) : (u, v, t) \in E$ and $(u, t) \notin \mathcal{T}$, by definition $\text{loss}(v, t)_i$ would be at least 1. So, in case $\text{loss}(v, t)_i = 0$, (v, t) is removed and along with that, the loss value of each vertex $u \in N(v)$ is increased by one as, after this iteration, the removal of u would make the edge (u, v, t) from covered to uncovered. The loss of all the vertices not in $N(v)$ do not change.
- 3) Suppose now that after the shrinking phase, there exists a vertex (v, t) in \mathcal{T} whose removal keeps \mathcal{T} a timeline cover. Supposing that the shrinking phase takes i iteration, from the assumption we can say that $\text{loss}(v, t)_i = 0$, by definition of loss . Since the loss value of any vertex (v, t) in \mathcal{T} does not decrease according to point 2 of this proof, the value of $\text{loss}(v, t)$ at the i^{th} iteration is at most 0, thus, since loss values are non-negative it is exactly 0. Therefore, (v, t) would have been removed at the i^{th} iteration. This complete the proof by contradiction. \blacktriangleleft

C SelectRndVertex Quality: Proof

Theorem 5 states that with $k \geq 50$, the probability that the SelectRndVertex algorithm, outlined in Algorithm 3, choose a vertex whose loss value is not grater than 90% vertices in \mathcal{T} is grater or equal to 0.9948.

Proof. Consider a real number $\rho \in (0, 1)$, the probability of the event $E = \{\text{the loss value of the element chosen by the SelectRndVertex algorithm is not greater than } \rho|\mathcal{T}| \text{ elements in the set } \mathcal{T}\}$ is $Pr(E) \geq 1 - (\frac{\rho|\mathcal{T}|-1}{|\mathcal{T}|})^k > 1 - \rho^k$.

Indeed, $\rho|\mathcal{T}|$ is the number of elements in the ρ fraction of \mathcal{T} with the worst losses (i.e., higher loss). Thus, considering that each random selection from \mathcal{T} is independent, $\frac{\rho|\mathcal{T}|}{|\mathcal{T}|}$ represents the chance of selecting one of the worst $\rho|\mathcal{T}|$ elements in one selection. It follows that $\frac{\rho|\mathcal{T}|-1}{|\mathcal{T}|}$ is the probability of selecting an element whose loss value is higher or equal than $(1 - \rho)|\mathcal{T}|$ elements. After k iterations, $(\frac{\rho|\mathcal{T}|-1}{|\mathcal{T}|})^k$ is the probability that all the iterations randomly select elements inside of the worst elements set. From hence $1 - (\frac{\rho|\mathcal{T}|-1}{|\mathcal{T}|})^k > 1 - \rho^k$ is the probability that at least one of the k iterations selects an element from within the top elements set.

This means that for $k = 50$, the probability that the SelectRndVertex algorithm choose a vertex whose loss value is not greater than $\rho = 90\%$ vertices in \mathcal{T} is $Pr(E) \geq 1 - 0.9^{50} > 0.9948$, where we consider the probability of event E to be greater or equal to the computed value, since there might be the case that more than one elements in those $\rho|\mathcal{T}|$ elements have the same loss value, which is the minimum among loss values of all the $\rho|\mathcal{T}|$ element. \blacktriangleleft