Algorithms and Complexity for Temporal Graph Optimization and Design

KLOBAS, NINA

Algorithms and Complexity for Temporal Graph Optimization and Design

Nina Klobas

Supervisor: Dr. George B. Mertzios

Second supervisor: Professor Matthew Johnson

A PhD Thesis presented for the degree of
Doctor of Philosophy

Department of Computer Science
Durham University
United Kingdom
May 2024

Dedication

To Tina and Katja.

Abstract

In this dissertation, we focus on graphs with an added time dimension, called temporal graphs. Temporal graphs naturally model graphs whose underlying topology changes over time. Our aim is to extend classical graph concepts with a temporal dimension and contribute to the algorithmic developments in temporal graphs. We achieve this by presenting an in-depth study of four different problems.

We start with the problem of finding temporally disjoint paths or walks between a given set of vertices, where two paths or walks are temporally disjoint if they do not intersect at any point in time. We focus on the special cases, where the underlying graph of the given temporal graph is either a tree or a path. While this problem proves to be computationally demanding in general, we identify specific cases where its complexity is more manageable.

Our research extends to the Temporal Vertex Cover (TVC) and Sliding-Window Temporal Vertex Cover (Δ-TVC) problems, natural extensions of the classic Vertex Cover problem. Interestingly, Δ-TVC proves to be computationally challenging already on simple graphs like paths or cycles, presenting a sharp contrast to the more tractable TVC in similar scenarios. We introduce various algorithms to solve these problems in specific types of temporal graphs.

Shifting our focus, we explore temporal design problems for ensuring connectivity in undirected temporally connected graphs. The core objective is to minimize the number of time-labels added to the edges while preserving temporal connectivity. We present scenarios where this task is computationally challenging and others where it demonstrates more tractable behavior, which leads us to investigate the complex connection between time and structure in these graphs.

Our final problem explores temporal graph realization, specifically in constructing periodic temporal graphs with fastest paths matching prescribed time durations between vertices. We show that while this problem is generally hard, it becomes more manageable when the underlying structure resembles a tree.

Acknowledgements

First and foremost I would like to thank my supervisor George Mertzios for all the time, hard work and long meetings he invested in me. I have learned a lot during the last four years with you, thank you.

I am especially grateful also to all of my collaborators: Thekla Hamm, Hendrik Molter, Rolf Niedermeier, Paul G. Spirakis and Philipp Zschoche. Working alongside such exceptional researchers has been a privilege. I appreciate the opportunity to have learned from each and every one of you.

My time in Durham would not have been so enjoyable without the support of my family and friends (some "old" ones and many "new" ones). To all of you, I express my love and appreciation for sharing this experience with me.

Contents

List of Figures

List of Tables

Declaration

The work presented in this dissertation is based on research conducted from October 2020 to March 2024, carried out at the Department of Computer Science, Durham University, United Kingdom in the ACiD (Algorithms and Complexity in Durham) research group, under the supervision of George B. Mertzios. Many of the results shared in this thesis are a product of published papers that were prepared in collaboration with my coauthors, in alphabetical order: Thekla Hamm, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Paul G. Spirakis and Philipp Zschoche.

Each chapter of this dissertation is based on its related paper. In collaborative works, it is common for authors to contribute unequally to various parts. Therefore, this thesis may omit certain results presented in our papers, focusing primarily on areas where I made significant contributions. For parts where my contribution might be less substantial, I briefly outline the findings for completeness and explicitly state why detailed proofs are not included. Each chapter begins with a clear explanation of the aspects of the project that are omitted.

CHAPTER 1

Introduction

The fundamental concept of a graph, simply a "collection of dots and lines", represents a basic yet powerful mathematical abstraction. In this abstraction, each dot represents a specific object, and a line between two dots represents a certain relation between them. Formally, a (simple) graph is a pair, consisting of a set of objects and a set of binary relations between them. We usually denote the graph as $G = (V, E)$, where V represents the set of vertices and $E \subseteq \binom{V}{2}$ the set of edges. Despite their apparent simplicity, graphs provide a robust framework for modelling a diverse range of everyday problems. This versatility has played a big role in the fast development of the study of graphs, known as Graph Theory, into a discipline with a rich history. For the birth of this discipline, we usually consider the year 1736 with the Köningsberg bridge problem. The city of Köningsberg (now Kaliningrad) was situated on both sides of the river, with two large islands in the middle. All of the parts of the city were connected to each other by seven bridges. The challenge posed was to find a path that would traverse each bridge exactly once. Leonhard Euler ingeniously addressed this problem by introducing an abstract mathematical object that later became known as a graph. He replaced each landmass with a vertex, and each bridge with an edge. By doing so Euler transformed the problem from finding the desired path in the city to finding a path on the graph, that would use each edge exactly once. After analyzing the properties of the derived graph, Euler concluded that no such path exists. This seminal work paved the way for centuries of research on graphs, inspiring many researchers to explore and introduce various properties and concepts in the field.

The simple definition of graphs, while straightforward, often proves insufficient and has to be extended to be used for modelling and solving more advanced problems. Take, for instance, the task of finding an optimum (shortest/fastest) path between two points in a city – a task we would normally use a GPS service like Google Maps to solve. In this scenario, we can transform the city map into a graph, similar to Euler's approach. We set each point of interest, along with every intersection, to be a vertex, and each road between two such points to be an edge between the corresponding vertices. However, a brief investigation reveals that the constructed graph lacks sufficient information to determine an optimal path accurately. This is true because the streets in cities are normally of different lengths. Therefore, an optimal path from point A to point B may naturally traverse many points/streets, yet it could still be better (shorter/faster) than an alternative route that traverses only a few of them (using fewer edges in our derived graph). To accurately determine the optimal path in such cases, we need to have more information in our graph. We achieve this by assigning extra values to the edges, representing street lengths (or traversal times). With this modification, we can now determine the optimal path. The resulting modified graph belongs to a special class of graphs called (edge) weighted graphs.

Another example of a modified definition of graphs occurs when the relations can involve more than two vertices each. In one scenario, $k \geq 2$ elements can be involved in one relation, resulting in $E \subseteq \binom{V}{k}$. Such graphs are referred to as hypergraphs. In an alternative scenario, the binary relation on elements of V is not symmetric, implying that the order of elements is important. In this case, if $(u, v) \in E$, it does not necessarily imply that $(v, u) \in E$. Here, we are dealing with directed graphs. It turns out that there exist various extensions of the definition of a graph that take into account a range of different structures and characteristics.

In this dissertation, our focus is on studying graphs that extend the basic definition by incorporating a time dimension. We refer to such graphs as *temporal graphs*. Within this framework, the set of vertices remains static, while the set of edges changes over time. Building upon the foundational work of Kempe et al. [84], we adopt the following simple and natural model for them.

Definition 1.0.1 (Temporal Graph)**.** *A* temporal graph *$\mathcal{G}$ is a pair (G, λ), where $G = (V, E)$ is an underlying (static) graph and $\lambda : E \to 2^{\mathbb{N}}$ is a* time-labeling *function which assigns to every edge of G a set of discrete-time labels.*

Due to their relevance and applicability in many areas, temporal graphs have been studied from various perspectives and under different names such as *dynamic* [26, 61], *evolving* [22, 32, 50], *time-varying* [1, 51, 122], and *graphs over time* [93]. For a comprehensive overview of the existing models and results on temporal graphs from a distributed computing perspective see the surveys [24–26, 101]. There were some previous attempts to incorporate a different perspective of time to the graph, for example, see Orlin [109, 110] or Berman [18], but these models are different from the one considered in our study.

While time flows linearly and is uniquely defined for everyone in the world, we have all experienced its complex nature. As a simple example, consider how five minutes can sometimes feel extremely long (imagine holding your breath for this amount of time), and conversely, can pass in a blink of an eye (especially in the morning, after pressing that snooze button on your alarm). Given the complicated nature of time, it is no surprise that introducing the time dimension increases the difficulty and complexity of different problems in graphs. The first obstacle already arises when attempting to extend/lift some basic definitions from static graphs to temporal graphs. For instance, a (natural) definition of a path in temporal graphs is a sequence of edges, where the edges must form a path in the underlying graph, and the labels have to be non-decreasing, reflecting the inability to travel backwards in time. Now, the interesting part arises when we aim to compute the "best" temporal path from a vertex u to a vertex v, where here "best" can be appropriately defined. In a simple, static graph, the optimum (shortest) path between vertices u and v has the minimum number of edges we need to traverse to reach v from u. In a temporal graph, we can consider (at least) three different analogues for the same problem. The path can be again interpreted as the path that uses the smallest number of edges to reach v from u (in this case we call it the *shortest* (u, v)*-temporal path*), it can be the path that, from the moment it starts at u uses the least amount of time to reach v (we call it the *fastest* (u, v)*-temporal path*) or it can be a path that finishes at the earliest possible time (we call it the *foremost* (u, v)*-temporal path*). Each of the above definitions is interesting and worth exploring for its own merits.

Lifting the simple definition of a shortest path to temporal graphs was already a three-step process. Coming up with the correct temporal problem can be even more complex. For instance, consider the concept of a clique. In static graphs, a clique is a set of vertices where each member of the set is connected to all other members. Extending the definition of a clique to the temporal graph setting can be approached in various ways. We present the following three variations. Firstly, we may require at least one time-edge between any two vertices in the clique, indicating that at least at one point in time, two vertices are connected. In a second variant, any two vertices in the clique must be connected at all time-steps. While both of the above definitions are valid, the latter one seems a bit too restrictive, and the former is not restrictive enough. This brings us to the third definition, utilizing *sliding time windows* as follows: for any two vertices in the clique, there must exist at least one edge in every Δ-consecutive units of time. This definition was first presented in the work of Virad et al. [126], where the authors studied the contact patterns among high-school students with the aim of determining groups of students that were interacting more frequently. In this example, expecting two students to be in constant interaction (i. e., at every time the measurement happened) is a bit too restrictive, as students naturally spend time apart during different classes. Conversely, expecting any two to interact at least once is clearly not restrictive enough. Following the initial work of Virad et al. many other graph problems were introduced on temporal graphs with the use of sliding time windows.

In the research area of temporal graph theory, we normally categorize problems into two distinct classes: those that are path-related and those that are non-path-related. Within both categories, there exists an extensive body of literature addressing various challenges and complexities. Some interesting path-related problems deal with temporal paths, temporal analogues of distance, reachability, exploration and centrality [3, 4, 27, 41, 46, 76, 86, 98, 102, 132]. On the other hand, non-path-related problems study temporal cliques, cluster editing problem on temporal graphs, temporal vertex cover, temporal graph coloring, temporal matching, and orientations of temporal graphs [5, 16, 28, 60, 78, 99, 100, 126, 131].

The main goal of this dissertation is to contribute to the development of algorithmic temporal graph theory, similar to the well-established algorithmic graph theory on static graphs. This objective is achieved through an in-depth analysis of four different problems, covering both path-related topics (covered in Chapters 3, 5 and 6), and non-path-related topics (covered in Chapter 4).

1.1 Short overview of the thesis contributions

The organization of this thesis is as follows. In Chapter 2, we begin by presenting fundamental definitions and notations from graph theory, temporal graph theory, parameterized algorithms, and approximation algorithms. In Chapters 3 to 6 we focus on the study of four different problems on temporal graphs. In the following, we give a brief summary of the contributions made by each chapter.

Chapter 3: **Interference-Free Walks in Time: Temporally Disjoint Paths.** In this chapter, we study the problem of finding disjoint temporal paths or walks that connect given pairs of source-sink vertices. In our setting two temporal paths (or walks) Q_1, Q_2 are considered disjoint if they do not intersect at any vertex or edge in a given time. This means that both Q_1 and Q_2 are allowed to visit the same vertex or edge, as long as they do not visit it at the same time step. We focus on the special cases where the underlying graph of the given temporal graph is either a tree or a path. It turns out that the problem of finding disjoint paths and walks in this setting is NP-hard. Moreover, we present an FPT algorithm for the case of finding disjoint temporal paths, where the parameter is the number of source-sink pairs. On the positive side, when restricting the problem even further – namely the underlying graph of the given temporal graph is a path P, and the source-sink vertices we want to connect consist only of the endpoints of P – then there is a polynomial-time algorithm solving it.

Chapter 4: **The Complexity of Temporal Vertex Cover in Small-Degree Graphs**. In their work Akrida et al. [5] introduced Sliding-Window Temporal Vertex Cover (or Δ-TVC for time-windows of a fixed-length Δ), as a natural extension of the well-known Vertex Cover problem on static graphs. Given a temporal graph (G, λ), and any $\Delta \in \mathbb{N}$ the aim of Δ-TVC is to find a set of vertex appearances S that cover every edge at least once at every Δ consecutive time-steps, while minimizing the size of S. Here a vertex appearance is a pair (v, t) for some vertex v and $t \in \{1, 2, \ldots, T\}$. The results we present in this chapter are built upon the work of Akrida et al. [5]. We show that for any $\Delta \geq 2$, Δ-TVC is NP-hard already when the underlying graph of the input temporal graph is a path or cycle, and provide a *Polynomial-Time Approximation Scheme* (PTAS) for it. We present also an exact algorithm for Δ-TVC with exponential running time dependency on the number of edges of the underlying graph. This algorithm is then used as a subroutine in the polynomial-time $(d-1)$-approximation algorithm, where d is the maximum vertex degree at any time-step of the input temporal graph. This result improves the d-approximation algorithm proposed by Akrida et al. We finish with presenting a fixed-parameter tractable algorithm, with respect to the size of an optimum solution.

Chapter 5: **The Complexity of Computing Optimum Labelings for Temporal Connectivity**. In this chapter, we present *temporal design* problems of undirected temporally connected graphs. The basic setting of these optimization problems is as follows: given an undirected graph G, what is the smallest number $|\lambda|$ of time-labels that we need to assign to the edges of G such that (G, λ) is temporally connected (i. e., there is a temporal path among each pair of vertices)? We show that the unrestricted problem, called Min. Labeling (ML) can be solved in polynomial time. We define three additional variations. One, where the input consists also of an upper bound of the allowed *age* (i. e., maximum label) of the obtained temporal graph (G, λ), we call this problem Min. Aged Labeling (MAL). And the next two where we consider problem variations with the aim of having temporal paths only between pairs of *terminals* that lie in a subset $R \subseteq V$. Similarly, in one problem there is no restriction on the largest label used - we call this problem Min. Steiner Labeling (MSL) - and in the second problem, the age restriction is used (i. e., the largest label we are allowed to use is bounded) - we call this version Min. Aged Steiner Labeling (MASL). We show that MAL is NP-hard when the required maximum age is equal to the diameter d_G of the input static graph G. We then go on to prove that MSL is NP-hard and provide a fixed-parameter tractable algorithm for it, with respect to the size of the labelling and the number of terminals. Lastly, we establish that MASL is W[1]-hard with respect to the number of terminals.

Chapter 6: **Realizing Temporal Graphs From Fastest Travel Times**. The (static) *graph realization* problem with respect to a graph property $\mathcal{P}$ is to find a graph that satisfies $\mathcal{P}$ or to decide that no such graph exists. In the simplest version of a (static) graph realization problem with respect to vertex distances, we are given a symmetric $n \times n$ matrix D and we are looking for an n-vertex undirected and unweighted graph G such that $D_{i,j}$ equals the distance between vertices v_i and v_j in G. We try to extend the idea of the graph realization problem with respect to vertex distances to the context of temporal graphs. We focus our study on *periodic* temporal graphs, i. e., temporal graphs in which the temporal availability of each edge of the underlying graph is periodic. More precisely, each edge $e \in E(G)$ is labeled with exactly one label ℓ_e from the set $\{1, 2, \ldots, \Delta\}$, which implies that edge e appears at times $\ell_e, \ell_e + \Delta, \ell_e + 2\Delta, \ell_e + 3\Delta, \ldots$ We define the problem of Simple periodic Temporal Graph Realization (Simple TGR) that is given as an input an $n \times n$ matrix D, together with a positive integer Δ, and asks if there exists a temporal graph $\mathcal{G} = (G, \lambda)$, where λ assigns one label to each edge of G, such that the resulting Δ-periodic temporal graph admits a fastest temporal path from v_i to v_j that is of duration $D_{i,j}$. We prove that Simple TGR is NP-hard already for a small constant Δ, and it is polynomial-time solvable if the underlying graph G is a tree. We then show that Simple TGR is W[1]-hard when parameterized by the feedback vertex number of the underlying graph, and present a fixed-parameter tractable algorithm for it, with respect to the feedback edge number of the underlying graph.

1.1.1 Publications

The work presented in this thesis represents a small yet noteworthy contribution to the field of temporal graphs. All of this research is an outcome of different collaborations, and most of the results have already appeared in different conferences and journals. The list of published results is presented in Table 1.1.

Table 1.1: List of publications this work is based on.

Title	**Authors**	**Place and Year of Publication**
Interference-free walks in time: Temporally disjoint paths	Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Philipp Zschoche	Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI) 2021 [85] Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) 2023 [86]
The Complexity of Temporal Vertex Cover in Small-Degree Graphs	Thekla Hamm, Nina Klobas, George B. Mertzios, Paul G. Spirakis	Proceedings of the 36th Conference on Artificial Intelligence (AAAI) 2022 [73] ArXiv version [74] Under the submission to the Journal of Artificial Intelligence Research (JAIR)

Table 1.1: List of publications this work is based on.

Title	**Authors**	**Place and Year of Publication**
The Complexity of Computing Optimum Labelling for Temporal Connectivity	Nina Klobas, George B. Mertzios, Hendrik Molter, Paul G. Spirakis	Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS) 2022 [87] ArXiv version [88] Under submission to the Journal of Computer and System Sciences (JCSS)
Realizing Temporal Graphs From Fastest Travel Times	Nina Klobas, George B. Mertzios, Hendrik Molter, Paul G. Spirakis	Proceedings of the 2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND) 2024 [90] ArXiv version [89]

CHAPTER 2

Definitions

In this chapter, we present all the concepts, notations, and terminology that we use throughout the thesis. Each chapter also has a separate preliminaries section where we introduce any chapter-specific concepts and notations.

With $\mathbb{N}$ and $\mathbb{N}_0$ we denote the natural numbers excluding and including 0, respectively. An interval on $\mathbb{N}_0$ from a to b is denoted by $[a, b] \coloneqq \{i \in \mathbb{N}_0 \mid a \leq i \leq b\}$ and $[a] \coloneqq [1, a]$.

2.1 Graphs

In this section, we present several graph-related definitions, that are necessary to understand the work of this thesis. Since the scope of graph theory is really broad, we present just a small part of the concepts. For a more comprehensive understanding of the field see [21, 127].

Definition 2.1.1. *A* graph *G is an ordered pair (V, E), consisting of the set of vertices $V(G) = V$ and the set of edges $E(G) = E \subseteq \binom{V}{2}$. If both $|V|$ and $|E|$ are finite, then we say that G is a* finite *graph. Two vertices u, v are adjacent if there exists an edge $e = \{u, v\} \in E$ between them. Two edges e_1, e_2 are adjacent if they* *are incident to the same vertex. A* self-loop *is an edge $\{u, v\}$ for which $u = v$. If E is a multi-set, meaning there are edges $e_1, e_2 \in E$ such that $e_1 = e_2$, then we say that e_1 and e_2 are* parallel edges*. A graph without self-loops and parallel edges is called a* simple *graph.*

A directed graph *or* digraph *D is an ordered pair (V, A), consisting of the set of vertices $V(D) = V$ and the set of arcs $A(D) = A$ of ordered pairs of elements in V. An arc (u, v) represents a directed edge oriented from u to v.*

Throughout this work, we are usually concerned only with finite, simple, undirected graphs. Therefore, unless specified otherwise, whenever we use the term graph we mean a finite, simple, undirected graph. For the sake of simplicity, we mostly drop the set notation from edge $e = \{u, v\}$ and use $e = uv$ instead.

Definition 2.1.2. *The number of edges incident with a vertex v, in a graph G is called the degree (or valence) of v, and is denoted by $deg_G(v)$.*

If every vertex in a graph G is of degree k, then we say that G is a *k-regular graph.*

Definition 2.1.3. *A* walk *of length k in a graph G is a sequence of vertices of G of the form: $S = (v_0, v_1, v_2, v_3, v_4, \ldots, v_k)$, where $e_i = v_{i-1}v_i \in E(G)$, for all $i \in [k]$. If the first and last vertex of the walk are the same, then the walk is called* closed*, otherwise, it is* open*. If all of the edges between consecutive vertices of S are distinct, the sequence S is called a* trail*, in the case when also all vertices of S are distinct we call S a path. A* cycle *is a path together with an additional edge between the first and the last vertex.*

We say that a graph G is *connected*, if there exists a path between any two vertices $u, v \in V(G)$ and *disconnected* if it is not connected. Let G be a graph with a path P on vertices $(v_0, v_1, \ldots, v_k)$. Then we say that P has length k, which we denote as $d_G(P) = k$.

Definition 2.1.4. *A path Q between vertices $u, v \in V(G)$ is called a* shortest (u, v)-path *if for any other path Q' between the same pair of vertices u and v*

it holds that $d_G(Q) \leq d_G(Q')$. For vertices u, v, we say that they are at distance d in G, when $d_G(u, v) = d_G(Q) = d$. The diameter d_G of a graph G is the length of the longest shortest path among any two vertices in G. More precisely $d_G = \max_{u,v \in V(G)}\{d_G(u, v)\}$.

A path on n-vertices is denoted by P_n, and a cycle on n-vertices by C_n. If a graph G has no cycles, then it is called *acyclic*. If such a graph is also connected, we call it a *tree*.

Definition 2.1.5. *Let $G = (V, E)$ and $H = (V', E')$ be two graphs. Graph H is a subgraph of G (denoted $H \subseteq G$), if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, moreover in the case where $V(H) = V(G)$ graph H is called a spanning subgraph of G. For a set $S \subseteq V$, $G[S]$ is the subgraph of G induced by S whose vertex set is S and whose edge set consists of all of the edges in E that have both endpoints in S.*

Given a set of objects O we define an *intersection graph* G_O, with a vertex for each object in O and an edge between vertices whose corresponding objects intersect. An example of an intersection graph is a *unit interval graph*, where the set of objects consists of intervals on the real line, that are of unit length.

2.1.1 Temporal graphs

In this section, we introduce a range of notation and terminology that is connected to temporal graphs. Although we already presented the full definition of temporal graphs in the introduction (see Definition 1.0.1) we restate it here again.

Definition 2.1.6. *A* temporal graph *$\mathcal{G}$ is a pair (G, λ), where $G = (V, E)$ is an underlying (static) graph and $\lambda : E \to 2^{\mathbb{N}}$ is a* time-labeling *function which assigns to every edge of G a set of discrete-time labels.*

The maximum value over all labels is called the *lifetime* T of $\mathcal{G}$, which can be finite or infinite. These time-labels indicate the discrete points in time, where each edge is present. For every $v \in V$ and every time $t \in [T]$, we denote the *appearance of vertex* v at time t by the pair (v, t) and the *edge appearance (or time-edge or temporal-edge)* of e at time t by (e, t). With $E_1, E_2, \ldots, E_T$ we denote the temporal edges that appear at time $1, 2, \dots, T$, respectively. Note that $E = \{e \mid (e, t) \in E_t, \text{ for some } t\}$. The graph $G_t = (V, E_t)$ is called the t-th *snapshot/layer* of $\mathcal{G}$.

One of the most central notions in temporal graphs is that of a temporal or time-respecting path which is motivated by the fact that, due to causality, entities and information in temporal graphs can "flow" only along sequences of edges whose time-labels are strictly increasing, or at least non-decreasing.

Definition 2.1.7. *Let $\mathcal{G} = (G, \lambda)$ be a temporal graph. A (u, v)-temporal path (or temporal path from u to v) in $\mathcal{G}$ is a sequence $\mathcal{P} = ((u = v_0 v_1, t_1), (v_1 v_2, t_2), \dots, (v_{k-1} v_k = v, t_k))$ where $P = (u = v_0, v_1, \dots, v_k = v)$ is a path in the underlying graph G, for all $i \in [k]$ it holds that $t_i \in \lambda(v_{i-1} v_i)$ and $t_1 \leq t_2 \leq \cdots \leq t_k$. In the case with $t_1 < t_2 < \cdots < t_k$ we say that $\mathcal{P}$ is a strict temporal path. The length of $\mathcal{P}$ is the same as the length of its static path P, the duration of $\mathcal{P}$ is $t_k - t_1 + 1$, and the arrival (or finishing) time of $\mathcal{P}$ is t_k. If P is a walk in the underlying graph G, then $\mathcal{P}$ in the temporal graph $\mathcal{G}$ is called a temporal walk.*

A shortest *(u, v)-temporal path is a temporal path from u to v that uses the smallest number of edges. A* fastest *(u, v)-temporal path is a temporal path from u to v with the smallest duration. A* foremost *(u, v)-temporal path is a temporal path from u to v with the earliest arrival time.*

Given a temporal graph $\mathcal{G} = (G, \lambda)$, there can be multiple temporal paths $\mathcal{P}$ traversing the vertices of a static path P in G. This makes the choice of labeling $\lambda' \subseteq \lambda$ for the edges of P important. To simplify notation in this thesis, there will be instances where we refer to P as a temporal path, when in fact we mean that the pair (P, λ') is a temporal path. In these instances, it is implied that the temporal path traverses the vertices of P, using its edges at times determined by λ'. Here, λ' is a subset of λ that makes (P, λ') either the fastest, the foremost, or a valid temporal path - this will be easily inferred from the context.

When devising algorithms for temporal graphs it is important to determine what is the *size* of the temporal graph $\mathcal{G} = (G, \lambda)$. The size depends on the *encoding size* of λ, which can be either the total number of the time-labels over all edges, i. e., $|\mathcal{G}| = |V| + \sum_{i=1}^{T} |E_i|$, or the length of a suitable succinct representation of λ.

Throughout the thesis, we refer to a temporal graph (G, λ) whose underlying graph G is a path, as a *temporal line graph*. This is in contrast to a temporal path (P, λ) in a temporal graph (H, λ), where P is a path in the graph H.

2.2 Parameterized complexity

In this section, we provide a short introduction of some basic definitions and notations from parameterized complexity theory. For a deeper overview of the research area consider [34, 38].

Definition 2.2.1. *Let* Σ *denote a finite alphabet. A* parameterized problem $L \subseteq \{(x, k) \in \Sigma^* \times \mathbb{N}_0\}$ *is a subset of all instances* (x, k) *from* $\Sigma^* \times \mathbb{N}_0$*, where* k *denotes the* parameter*.*

For example, an instance in the case of Vertex cover parameterized by the solution size k is the pair (G, k), where k is a positive integer and $G = (V, E)$ is an undirected graph encoded as a string over the alphabet Σ. Therefore (G, k) belongs to the Vertex cover parameterized language if and only if there exists a set $C \subseteq V$ of size k, such that C is a vertex cover of G. We define the size of the instance (x, k) of some parameterized problem as $|x| + k$.

Definition 2.2.2. *A parameterized problem* L *is* fixed-parameter tractable *(*FPT*) if there exists a* fixed parameterized algorithm $\mathcal{A}$ *and a computable function* $f : \mathbb{N}_0 \mapsto \mathbb{N}_0$ *for which the algorithm correctly decides in* $f(k) \cdot |x|^{O(1)}$ *time whether an instance* (x, k) *is in* L*.*

An example of a problem in the class FPT is the Vertex cover parameterized with respect to the solution size. Let (G, k) be an instance of Vertex cover, where k is the solution size and $G = (V, E)$ with $|V| = n$. The naive brute-force approach that calculates all possible k-subsets of vertices (there are $\binom{n}{k}$ such sets) and checks for each of them if it forms a vertex cover runs in time $O(n^{k+2})$, which is too slow, as by the definition of FPT we want the exponent on n in the running time to be a constant, independent of k. Therefore, a little smarter approach is needed. By the definition of Vertex cover, an edge is covered when at least one of its endpoints is in the cover. So we can use this fact to devise a recursive algorithm, that picks an uncovered edge, selects one of its endpoints to be in the vertex cover and continues to cover other uncovered edges. If, after k steps the set of selected vertices does not cover all the edges, it is not a vertex cover and the algorithm has to backtrack. The algorithm starts each step by picking an uncovered edge and putting one of its endpoint to the vertex cover. Since each edge has 2 endpoints there are 2 different possible outcomes of this step, for a selected edge. This step is performed at most k times, which results in at most 2^k different sets of vertices. The only thing left to do is to check if a set of selected vertices is a vertex cover. If there is at least one set that is a vertex cover the algorithm returns YES, otherwise NO. The running time of this algorithm is $O(2^k n^2)$, which satisfies the conditions in the definition of FPT.

As we observed from the running time of the naive approach, sometimes there are parameterized algorithms that run in a time greater than in the case of FPT. Therefore we need to define also the following class.

Definition 2.2.3. *A parameterized problem L is* slice-wise polynomial *(*XP*) if there exists an algorithm $\mathcal{A}$ and two computable functions $f, g : \mathbb{N}_0 \to \mathbb{N}_0$ for which the algorithm correctly decides in $f(k) \cdot |x|^{g(k)}$ time, if an instance (x, k) is in L.*

There are cases when there is no polynomial-time algorithm even when k is a constant. An example of this kind of problem is deciding whether a graph can be properly colored with 3 colors. These problems are in the following class.

Definition 2.2.4. *A parameterized problem L is* para-NP-hard *if already for some constant value of the parameter k the problem is* NP*-hard.*

In the case of NP-hard problems there is a notion of polynomial-time reductions that is usually used in hardness proofs. Similar to that, there is an analogous notion of a reduction for parameterized problems that preserves the fixed-parameter tractability. Let us define it.

Definition 2.2.5. *Let L_1, L_2 be two parameterized problems. A* parameterized reduction *(called also* FPT-reduction*) from L_1 to L_2 is an algorithm $\mathcal{A}$ that for a given instance $(x, k) \in L_1$ outputs an instance $(x', k') \in L_2$ such that:*

1. *(x, k) is a YES instance of L_1 if and only if (x', k') is a YES instance of L_2,*

2. *for some computable function g we have $k' \leq g(k)$ and*

3. *the running time of $\mathcal{A}$ is $f(k) \cdot |x|^{O(1)}$, for some computable function f.*

For these reductions the following holds.

Theorem 2.2.6 ([34], p. 424)**.** *If there is a parameterized reduction from L_1 to L_2 and L_2 is in* FPT *then also L_1 is in* FPT*.*

Comparing the definitions of an FPT-reduction to a known polynomial-time reduction of NP-hard problems we see that Item 2 in the definition of the former, gives some extra requirements. In particular, the new parameter has to be upper-bounded by a function of the parameter of the original instance. Therefore, not every NP-hardness reduction gives a parameterized reduction. For example, polynomial reduction from Independent Set to Vertex Cover, where (G, k) is a YES instance of the Independent Set if and only if $(G, |V| - k)$ is a YES instance of the Vertex Cover, does not fulfil the requirements of Item 2 and is therefore not an FPT-reduction.

Not all parameterized problems can be reduced to each other. In some cases, the one-way reduction (i. e., from L_1 to L_2) is known but the other way (i. e., L_2 to L_1) is not. This suggests that, unlike in the case of NP-complete problems, there is a hierarchy of hard parameterized problems. For example Independent Set and Vertex Cover are in different classes. Therefore Downey and Fellows [38] define the W-*hierarchy*.

The class FPT $=$ W$[0]$ and $W[i] \subseteq W[j]$ for all $i \leq j$. All the classes in the W-hierarchy are closed under FPT-reductions. If a parameterized problem L is W$[1]$-hard, then it is presumably not fixed-parameter tractable. In fact, it is expected that W$[i] \neq$ W$[i+1]$ for every $i \geq 0$.

Some known results of this hierarchy include:

- Independent Set is W$[1]$-hard,

- Dominating Set, Set Cover and Hitting Set are W$[2]$-hard.

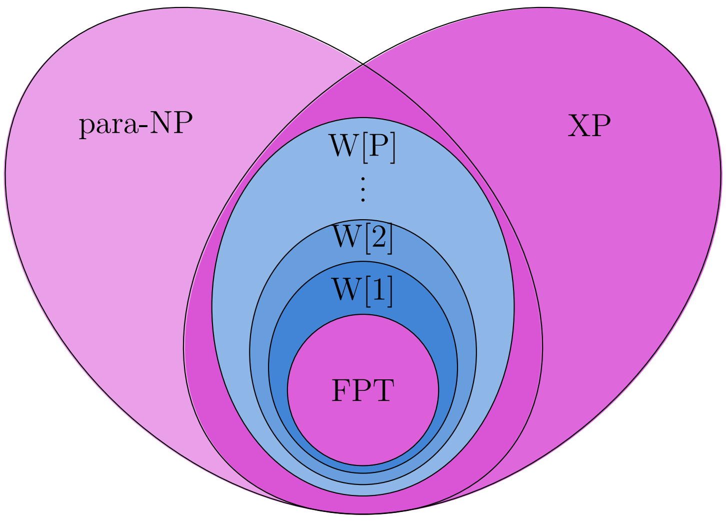


Figure 2.1: A visual representation outlining the relationships within complexity classes.

A diagram illustrating the relationships among all the above complexity classes is depicted in Figure 2.1.

2.3 Approximation algorithms

In various optimization problems, achieving an optimal solution in polynomial time is often impossible. However, by relaxing our demands concerning solution quality (optimality), we often find algorithms that not only run within a reasonable time-frame but also yield solutions reasonably close to the optimal. Such algorithms are known as approximation algorithms. In this chapter, we introduce basic notations and definitions from the field of approximation algorithms. For a more comprehensive overview, refer to [125, 128].

Definition 2.3.1. *Let X be a minimization/maximization problem and let $\alpha \geq 1$. An algorithm $\mathcal{A}$ is called an* α-approximation algorithm *for problem X, if for all instances I of X the algorithm calculates the solution $\mathcal{A}(I)$, that is at most α away from the optimum, i. e.,*

$$\max\left\{\frac{A(I)}{OPT(I)}, \frac{OPT(I)}{A(I)}\right\} \leq \alpha$$

where the $\frac{A(I)}{OPT(I)} \leq \alpha$ corresponds to a minimization problem and $\frac{OPT(I)}{A(I)} \leq \alpha$ to a maximization problem.

An easy example of an approximation algorithm is a 2-approximation algorithm for the vertex cover problem, that first calculates a maximum matching (in polynomial-time) and puts in the vertex cover both endpoints of each edge in the matching.

The value α, from the definition of approximation algorithms, is viewed as a measure of the performance of the algorithm. The closer it is to 1 the better the approximation algorithm it becomes. In some cases, the bound can come arbitrarily close to 1, in other cases the lower bound is fixed and sometimes there exist no constant-factor approximation algorithms. In particular, the following classes have been identified.

1. Problems that are not α-approximable in polynomial-time, for any $\alpha > 0$, unless $\mathsf{P} = \mathsf{NP}$.

2. Constant factor approximable (APX) problems. These are the problems that can be approximated by some constant $\alpha > 0$.

3. Problems admitting a polynomial-time approximation scheme (PTAS). A PTAS algorithm is an algorithm that produces a solution which is within a factor $(1 + \epsilon)$ (or $(1 - \epsilon)$ in case of maximization problems) away from the optimum and runs in $O(|I|^{f(1/\epsilon)})$ time, where $|I|$ is the size of the instance of the problem.

4. Problems admitting a fully polynomial-time approximation scheme (FPTAS). An FPTAS algorithm, similarly with PTAS, produces a solution within a factor $(1+\epsilon)$ (or $(1-\epsilon)$) away from the optimum but runs in a time polynomial both in the input size and $1/\epsilon$, i. e., $O(|I|^{O(1)} \cdot (1/\epsilon)^{O(1)})$.

CHAPTER 3

Interference-Free Walks in Time: Temporally Disjoint Paths

This chapter is based on a joint work with George B. Mertzios, Hendrik Molter, Rolf Niedermeier and Philipp Zschoche. The preliminary results were presented in Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI) [85] and the full paper was published in the Journal of Autonomous Agents and Multi-Agent Systems [86].

In this thesis I focus only on the second part of the full paper [86], as my involvement in the results from the first part was not significant. However, for the sake of completeness, an overview of the key results from the first part is provided. I believe that the inclusion of these results provides a better understanding and presents a clearer picture of the complexity of the addressed problem.

3.1 Introduction

Computing (vertex-)disjoint paths in a graph is a cornerstone problem of algorithmic graph theory and many applied network problems. It was among the early problems that were shown to be NP-complete [81]. One of the deepest achievements in discrete mathematics, graph minor theory [106,107], as well as the development of the theory

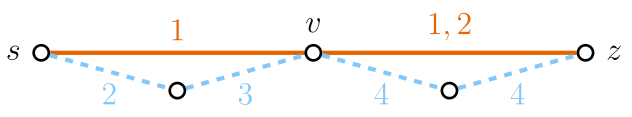


Figure 3.1: A temporal graph where a label of an edge reflects at which time it is available. There are two temporally disjoint (s, z)-paths P_1 and P_2, where P_1 uses the solid (orange) edges and P_2 the dashed (blue) edges. Here, P_1 visits v before P_2.

of parameterized complexity analysis [39] are tightly connected to it. The problem is known to be solvable in quadratic time if the number of paths is constant, that is, it is fixed-parameter tractable when parameterized by the number of paths [83]. Besides being of fundamental interest in (algorithmic) graph theory, finding disjoint paths has many applications and there exist numerous variations of the problem. In AI and robotics scenarios, for instance, multi-agent path finding is an intensively studied, closely related problem [119, 120].

Coming from the graph-algorithmic side, we propose a new view on finding disjoint paths (and walks), that is, we place the problem into the world of temporal graphs. In our model, we consider two paths (or walks) to be disjoint if they do not use the same vertex at the same point in time. Consider Figure 3.1 for an example. Moreover, the path finding also has to take into account that edges are not permanently available, reflecting dynamic aspects of many real-world scenarios such as routing in traffic or communication networks, or the very dynamic nature of social networks. We intend to initiate studies on this natural scenario. Doing so, we focus on two extreme cases for the underlying graphs, namely the (underlying) graph structure being completely unrestricted or being restricted to just a path graph. For these opposite poles, performing (parameterized) computational complexity studies, we present surprising discoveries. Before coming to these, we discuss (excerpts of) the large body of related work.

Related work As mentioned above, the literature on (static) disjoint paths and on multi-agent path finding is very rich. Hence, we only list a small fraction of the relevant related work. In the context of graph-algorithmic work, the polynomial-time (in-)approximability of the NP-hard maximization version has been studied [31]. Variants of the basic problem studied include bounds on the path length [62] or relaxing on the disjointness of paths [52, 53, 68, 123].

In directed graphs, finding two disjoint paths is already NP-hard [54], whereas in directed acyclic graphs the problem is solvable in polynomial time for every fixed number of paths [117].

As to multi-agent path finding, we remark that it has been intensively researched (with several possible definitions) in the last decade in the AI and robotics communities [7, 9, 118–120]. Timing issues (concurrency of moving agents) and the various objective functions of the agents play a fundamental role here; also a high variety of conflict scenarios is studied.[1] The scenario we study in this work can be interpreted as a basic variant of multi-agent path planning, now translated into the world of temporal graphs.

In algorithmic graph theory, edge-colored graphs have also been studied. Edge-colored graphs are essentially multilayer (or multiplex) graphs where the fundamental difference to temporal graphs is that there is no order on the graph snapshots (also referred to as layers). Here, path-finding scenarios are motivated, for example, by applications in social and optical (routing) networks [37, 114, 129].

Finally, as to temporal graphs, note that several prominent graph problems have been studied in this fairly new framework. In particular, another model of vertex-disjoint temporal paths [84], where two temporal paths are considered vertex-disjoint if they do not visit the same vertex. The problem of finding two such paths is NP-hard [84]. Note that the major difference to our model is that we allow two *temporally* disjoint paths to visit the same vertex as long as they do not both visit that vertex at the same *time*.

Our contributions.

When the studied temporal graph $\mathcal{G} = (G, \lambda)$ is a temporal line or a temporal tree (i.e., the underlying graph of $\mathcal{G}$ is a path or a tree, respectively), we show that the problem of finding disjoint temporal paths and walks is NP-hard. However, we also provide a fixed-parameter tractability result with respect to the number of paths.

[1] Also see the multi-agent path planning webpage: http://mapf.info/

Temp. Disjoint	Paths	Walks
temporal line or tree	NP-hard FPT wrt. $\|S\|$	FPT wrt. $\|S\|$ [91]
temporal line & S contains only pairs of extremal points	poly-time	
unrestricted underlying graph	NP-hard for $\|S\| = 2$	W[1]-hard wrt. $\|S\|$ XP wrt. $\|S\|$

Table 3.1: Overview of computational complexity of Temporally Disjoint Paths/Walks. Here, S is the multiset of source-sink pairs. Temporal line means that the underlying graph is a path.

For the special case where, in an input temporal line, the given multiset of source-sink pairs only contains pairs of the extremal points of the temporal line, we provide a polynomial-time algorithm.

Further contributions of the paper this chapter is based on are for temporal graphs where the underlying graph is unrestricted. In this case, finding walks instead of paths turns out to be computationally easier. More specifically, finding temporally disjoint walks is W[1]-hard with respect to the number of walks but can be solved in polynomial time if this number is constant (that is, in the language of parameterized algorithmics, there is an XP algorithm), whereas finding temporally disjoint paths already turns out to be NP-hard for two paths. All of our results are outlined in Table 3.1.

Further results. The work presented in this chapter inspired the research by Kunz et al. [91], where the authors provided an almost complete picture of the parameterized computational complexity of the presented problems, when structural graph parameters of the underlying graph combined with the number of source-sink pairs are considered. They show that the problem is in FPT when parameterized by the number of walks if the underlying graph is a path, which answered an open question left by our research. They prove the problem is NP-hard also for the temporal graphs when the underlying graph is a star, and show that in this case, the problem is W[1]-hard when parameterized by the number of vertices of the star graph. Furthermore, they explore the parameterized hardness of finding disjoint paths with respect to the combination of the number of disjoint paths and some extra graph-related parameters. Namely, they show that the problem is W[1]-hard when parameterized by the number of disjoint paths together with the vertex cover number of the underlying graph, and it is fixed-parameter tractable when parameterized by the feedback edge number combined with the number of paths.

3.2 Preliminaries and problem definition

Let $P = ((v_{i-1}v_i, t_i))_{i=1}^k$ be a temporal walk, where for all $i \in [k]$ we have that $\{v_{i-1}, v_i\} \in E_{t_i}$ and for all $i \in [k-1]$ we have that $t_i \leq t_{i+1}$. We say that P *visits* the vertices $V(P) \coloneqq \{v_i \mid i \in [0,k]\}$. In particular, P visits vertex v_i during the time interval $[t_i, t_{i+1}]$, for all $i \in [k-1]$. Furthermore, we say that P visits v_0 at time t_1 and P visits v_k at time t_k.

Given two temporal walks/paths P_1, P_2 we say that P_1 and P_2 *temporally intersect* if there exists a vertex v and two time intervals $[a_1, b_1], [a_2, b_2]$, where $[a_1, b_1] \cap [a_2, b_2] \neq \emptyset$, such that v is visited by P_1 during $[a_1, b_1]$ and by P_2 during $[a_2, b_2]$. Now, we can formally define our problem.

Temporally Disjoint Paths

Input: A temporal graph $\mathcal{G} = (V, (E_i)_{i\in[T]})$ and a multiset S of source-sink pairs containing elements from $V \times V$.

Question: Are there pairwise temporally non-intersecting temporal (s_i, z_i)-paths for all $(s_i, z_i) \in S$?

Analogously, Temporally Disjoint Walks gets the same input but asks whether there are pairwise temporally non-intersecting temporal (s_i, z_i)-walks for all $(s_i, z_i) \in S$. From the NP-hardness of Disjoint Paths [81], we immediately get the following.

Observation 3.2.1. *Temporally Disjoint Paths/Walks is NP-hard even if $T = 1$.*

Recall that a *foremost* temporal path from s to z starting at time t_0, in a given temporal graph $\mathcal{G}$, is a temporal path which starts at s not earlier than at time t_0 and arrives at z with the earliest possible arrival time. Similar to this we define an *always-foremost temporal path* $P = ((s = v_0, v_1, t_0), (v_1, v_2, t_2), \ldots, (v_{k-1}, v_k = z, t_k))$ from s to z as a temporal path where each prefix path is also a foremost temporal path. Namely, for every vertex v_i from the path P we have that the subpath from s to v_i forms a foremost temporal path in $\mathcal{G}$. A foremost temporal path between two vertices can be computed in linear $O\left(|V| + \sum_{i=1}^{T} |E_i|\right)$ time [130], by slightly modifying the algorithm the same result follows for the always-foremost temporal paths.

3.3 Temporal lines and trees

In this section, we investigate the computational complexity of Temporally Disjoint Paths/Walks for restricted classes of underlying graphs, in particular, *temporal lines* and *temporal trees*. The former are temporal graphs that have a path as an underlying graph and the latter are temporal graphs that have a tree as underlying graph. We first show that, to our surprise, the problem is NP-hard on temporal lines (and thus also on temporal trees). On the positive side, we show that, on temporal trees, Temporally Disjoint Paths is fixed-parameter tractable with respect to the number of source-sink pairs. If we further restrict all source-sink pairs to consist of the two end-points of the temporal line, however, then we obtain a polynomial-time algorithm.

Theorem 3.3.1. *Temporally Disjoint Paths/Walks is* NP*-hard even on a temporal line where all temporal paths are to the same direction.*

Proof. We present here a polynomial-time reduction for Temporally Disjoint Walks. Towards the end of this proof, we argue that the same reduction also works for Temporally Disjoint Paths. The reduction is done from Multicolored Independent Set on Unit Interval Graphs, which is known to be NP-complete [19, Lemma 2]. In this problem, the input is a unit interval graph $G = (V, E)$ with n vertices, where V consists of k subsets of independent vertices;

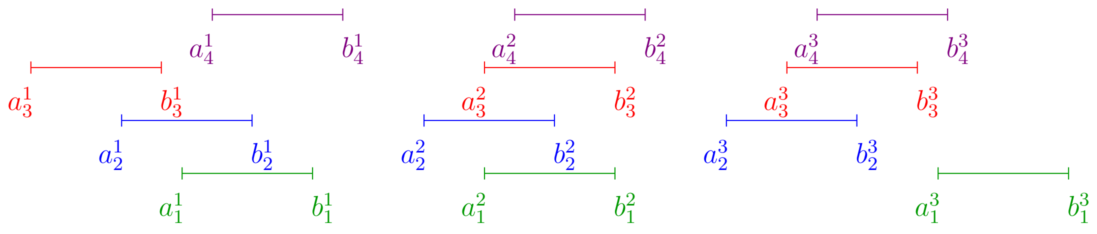
we interpret each of these subsets as a vertex color. The goal is to compute an independent set of size k in G which contains exactly one vertex from each color. By possibly slightly shifting the endpoints of the intervals in the given unit interval representation of G, we can assume without loss of generality that all endpoints of the intervals are distinct. Furthermore, we can assume without loss of generality that each interval endpoint is an integer between $k+1$ and $k+n^2$ (while all intervals still have the same length).

Construction From the given multi-colored unit intervals in G, we construct a temporal line $\mathcal{P}$ using the following procedure. Let $\{c_1, \ldots, c_k\}$ be the set of all colors of the intervals in G. First we fix an arbitrary linear ordering $c_1 < c_2 < \ldots < c_k$ of the k colors, and we add to the underlying path P of $\mathcal{P}$ two vertices v_i^1 and v_i^2, for every color c_i. We add to P also three basis vertices $v_\ell, v^\star, v_r$. The vertices of P are ordered starting from $v_1^1, v_2^1, \ldots, v_k^1$, followed by the basis vertices $v_\ell, v^\star, v_r$, and finishing with $v_1^2, v_2^2, \ldots, v_k^2$. At the end we have $P = (v_1^1, v_2^1, \ldots, v_k^1, v_\ell, v^\star, v_r, v_1^2, v_2^2, \ldots, v_k^2)$.

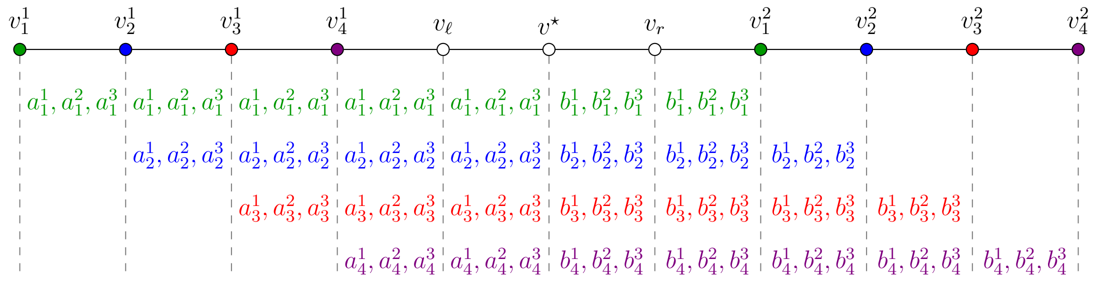
We construct the multiset S of source-sink pairs as follows. Let m_i be the number of intervals of color c_i. For every color c_i we add the pair (v_i^1, v_i^2) to S. We refer to this source-sink pair as "the verification source-sink pair for color c_i". Furthermore, we add $m_i - 1$ copies of the pair (v_i^1, v_ℓ) to S and we add $m_i - 1$ copies of the pair (v_r, v_i^2) to S. We call these $2m_i - 2$ source-sink pairs the "dummy source-sink pairs for color c_i".

To fully define the temporal line $\mathcal{P}$, we still need to add time labels to the edges of P. Denote by a_i^j and b_i^j the start and end points of the jth interval of color c_i. We set up the edge labels of path P from v_i^1 to v_i^2 as follows. To edge $\{v_s^1, v_{s+1}^1\}$ with $s \in [k-1]$, we add the labels a_i^j with $i \leq s$. To edges $\{v_k^1, v_\ell\}$ and $\{v_\ell, v^\star\}$, we add all labels a_i^j. To edge $\{v_s^2, v_{s+1}^2\}$ with $s \in [k-1]$, we add the labels b_i^j with $i > s$. To edges $\{v^\star, v_r\}$ and $\{v_r, v_1^2\}$, we add all labels b_i^j. See Figure 3.2 for an example. The construction can clearly be performed in polynomial time.

Correctness $(\Rightarrow)$: Assume there is a multicolored independent set $V' \subseteq V$ in G. Let $v_i \in V'$ be the vertex in the independent set with color c_i and



(a) An instance of the multicolored unit interval problem.



(b) Temporal graph constructed from the given multicolored unit intervals.

Figure 3.2: An example of the reduction described in the proof of Theorem 3.3.1.

let $[a_i^j, b_i^j]$ be the interval of v_i. Then for the verification source-sink pair of c_i we use the following temporal path: $((v_i^1, v_{i+1}^1, a_i^j), (v_{i+1}^1, v_{i+2}^1, a_i^j), \ldots, (v_{k-1}^1, v_k^1, a_i^j),$ $(v_k^1, v_\ell, a_i^j), (v_\ell, v^\star, a_i^j), (v^\star, v^r, b_i^j), (v_r, v_1^2, b_i^j), (v_1^2, v_2^2, b_i^j), \ldots, (v_{i-1}^2, v_i^2, b_i^j))$. For the dummy source-sink pairs (v_i^1, v_ℓ) of c_i we use the temporal paths $((v_i^1, v_{i+1}^1, a_i^{j'}), \ldots,$ $(v_{k-1}^1, v_k^1, a_i^{j'}), (v_k^1, v_\ell, a_i^{j'}))$ with $j' \neq j$. Note that there are exactly $m_i - 1$ pairwise different paths of this kind. Analogously, for the dummy source-sink pairs (v_r, v_i^2) of c_i we use the temporal paths $((v_r, v_1^2, b_i^{j'}), (v_1^2, v_2^2, b_i^{j'}), \ldots, (v_{i-1}^2, v_i^2, b_i^{j'}))$ with $j' \neq j$. It is easy to check that the temporal paths for the dummy source-sink pairs of all colors do not temporally intersect. Now assume, for the sake of contradiction, that temporal paths of two verification source-sink pairs of colors c_i and $c_{i'}$ temporally intersect. Then they have to intersect in $v^\star$, since this is the only vertex where the paths wait. By construction, the temporal path for the verification source-sink pairs of color c_i visits $v^\star$ during the interval $[a_i^j, b_i^j]$ and the verification source-sink pairs of color $c_{i'}$ visits $v^\star$ during the interval $[a_{i'}^{j'}, b_{i'}^{j'}]$. These two intervals correspond to the intervals of the vertices of colors c_i and $c_{i'}$ in the multicolored independent set V'. Hence, those intervals intersecting is a contradiction to the assumption that V'

is in fact an independent set.

($\Leftarrow$): Assume we have a set of pairwise temporally disjoint walks for all source-sink pairs in S. Note that all edges except $\{v_\ell, v^\star\}$ and $\{v^\star, v_r\}$ have as many time labels as temporal walks that need to go through them. Furthermore, note that $\{v_\ell, v^\star\}$ has the same labels as $\{v_k^1, v_\ell\}$ and $\{v^\star, v_r\}$ has the same labels as $\{v_r, v_1^2\}$. This, in particular, implies that all temporal walks are in fact paths since the only vertex that could be visited by a path for more than one time step is $v^\star$. Therefore, for every pair $(s, z) \in S$, no temporal path from s to z can ever stop and wait at any vertex different from $v^\star$. Furthermore, the only paths going through vertex $v^\star$ are the paths connecting vertices v_i^1 and v_i^2 (which correspond to color c_i); we will refer to this path as the color path of c_i. Consider color c_1 and its dummy source-sink pairs (v_1^1, v_ℓ). By construction, the edge $\{v_1^1, v_2^1\}$ has time labels corresponding to the start points a_1^j of intervals from the m_1 vertices of G that have color c_1. It follows that the temporal paths for these dummy source-sink pairs and the color path of c_1 use only time labels corresponding to the start points a_1^j of intervals from the m_1 vertices of G that have color c_1 until they are at v_ℓ or arrive at $v^\star$, respectively, since they cannot wait at any vertex. Now by induction, this holds for all other colors c_i and by an analogous argument, this also holds for the "second half". More specifically, we also have that temporal paths for the dummy source-sink pairs (v_r, v_i^2) as well as the "second part" of the color path of c_i use time labels corresponding to end points b_i^j of intervals from the vertices of G that have color c_i when going from v_r (respectively $v^\star$) to their corresponding destinations.

It follows that each color path can enter and leave vertex $v^\star$ only at the time corresponding to the start and end points of its color intervals. In any other case some of the other vertices are blocked, which prevents the completion of other temporal S-paths. Recall that intervals of the same color are non-overlapping. Hence, for every color path corresponding to a color c_i we can find one interval $[a_i^j, b_i^j]$ such that the color path visits $v^\star$ in an interval that includes $[a_i^j, b_i^j]$. Since the color paths are temporally non-intersecting, the vertices corresponding to the intervals form a multicolored independent set in G.

This completes the proof for the case of Temporally Disjoint Walks. As, in the constructed reduction, all walks are actually just paths, it follows that also Temporally Disjoint Paths is NP-hard. □

NP-hardness even in the case of temporal lines motivates to study the potential for parameterized tractability results. Next, we show fixed-parameter tractability of Temporally Disjoint Paths parameterized by the number $|S|$ of source-sink pairs if the underlying graph is a tree.

Theorem 3.3.2. *Temporally Disjoint Paths on temporal trees is fixed-parameter tractable when parameterized by $|S|$, as it can be solved in $O\left(|S|^{|S|} \cdot |\mathcal{G}|\right)$ time.*

Proof. Let $I = (\mathcal{G}, S)$ be an instance of Temporally Disjoint Paths, the underlying graph G being a tree and S consisting of k source-sink pairs $(s_1, z_1), \ldots, (s_k, z_k)$. We solve I using the following procedure.

We enumerate all possible permutations π of the k source-sink pairs (s_i, z_i). For every permutation π and for every $i = 1, 2, \ldots, k$, we compute the always-foremost temporal path $P_i(\pi)$ from s_i to z_i (i.e., a foremost temporal path where each prefix path is also a foremost temporal path). Let v_x be an arbitrary internal vertex of $P_i(\pi)$, and suppose that v_x is visited by $P_i(\pi)$ within the time interval $[a_x, b_x]$. Then, we mark all the edges that are incident to v_x with labels $\ell \leq b_x$, as these temporal edges cannot be used by any further temporal path $P_j(\pi)$, where $j > i$. We proceed by computing the always-foremost path $P_{i+1}(\pi)$ from s_{i+1} to z_{i+1} which only uses unmarked temporal edges. The permutation π leads to a feasible routing of the paths between the k source-sink pairs if and only if we can compute all these k always-foremost paths $P_1(\pi), P_2(\pi), \ldots, P_k(\pi)$ as described above.

During the above procedure we construct $O(k!) = O(k^{k-1})$ different permutations π. For every permutation we calculate $|S| = k$ always-foremost paths, each in $O(|V| + \sum_{i=1}^{T} |E_i|)$ time [130]. In total, all the above computations can be done in $O\left(|S|^{|S|} \cdot \left(|V| + \sum_{i=1}^{T} |E_i|\right)\right)$ time.

The correctness of the algorithm follows from the uniqueness of the paths P_i between source-sink pairs (s_i, z_i) in G, which implies that the intersection between two such paths $P_i \cap P_j$ is also uniquely determined. This means that the choice of the labels of the temporal (s_i, z_i)-path affect the choice of the labels of the temporal

(s_j, z_j)-path only on the edges that are incident to the vertices in the intersection $P_i \cap P_j$. We therefore need to determine which of the two paths has a priority and passes through the intersection first. Once the path of the highest priority is determined, we route that path so that it traverses the intersection at the earliest possible time leaving the most time for the second path (meaning we calculate the always-foremost temporal path). We then check if we can route the remaining path in such a way that it would be temporally disjoint with the first path. A single path P_i can intersect at most $k-1$ other P_j paths, therefore we need to check all possible orderings of priorities among them. Note also that the priority ordering is transitive. This follows from the fact that G is a tree. More specifically, for any three paths P_i, P_j, P_k in G we have that if P_i intersects with P_j and P_k then either $P_j \cap P_k = \emptyset$ or $P_i \cap P_j \cap P_k \neq \emptyset$. Which assures us that in a situation when the path P_i has a lower priority than P_k and a higher priority than P_j, the path P_k will have a higher priority than P_j. From the above, it also follows that all possible intersections of different source-sink paths in G follow the same priority ordering. Therefore, whenever a solution exists all paths follow a certain total order (permutation) and our algorithm will test it and find a solution. □

Finally, we show that we can solve Temporally Disjoint Paths/Walks in polynomial time if the underlying graph is a path and all source-sink pairs consist of the endpoints of that path.

Theorem 3.3.3. *Let $\mathcal{G}$ be a temporal line having $P = (v_0, v_1, v_2, \ldots, v_n)$ as its underlying path. If S contains k times the source-sink pair (v_0, v_n) and $\ell = |S| - k$ times the source-sink pair (v_n, v_0), then* Temporally Disjoint Paths/Walks *can be solved on $\mathcal{G}$ in $O(Tk\ell(k+\ell) \cdot |\mathcal{G}|)$ time.*

Proof. We first consider the problem version Temporally Disjoint Paths. Let $I = (\mathcal{P}, S)$ be an instance of Temporally Disjoint Paths, where $\mathcal{P}$ is a given temporal line with $P = (v_0, v_1, v_2, \ldots, v_n)$ as its underlying path. Assume that there have to be k (resp. $\ell = |S| - k$) temporally disjoint (v_0, v_n)- (resp. (v_n, v_0)-) paths in the output, i. e., they must have the orientation from v_0 to v_n, (resp. from v_n to v_0).

We solve the instance I using dynamic programming. The main idea is that, since all temporal paths start and end in endpoints of P, in any optimal solution, once a temporal path starts, it proceeds in the fastest possible way (without interfering with previously started paths). Therefore, assuming we start with (v_0, v_n)-temporal paths, we only need to find out *how many* (v_0, v_n)-temporal paths follow the starting path, after that *how many* (v_n, v_0)-temporal paths follow, then after that *how many* (v_0, v_n)-temporal paths follow, etc.

Let $0 \leq i \leq k$, $0 \leq j \leq \ell$, and $1 \leq t \leq T$. Then $L(i, j, t)$ denotes the earliest arrival time of $(k-i)+(\ell-j)$ temporally non-intersecting temporal paths with $k-i$ being (v_0, v_n)-temporal paths and $\ell - j$ being (v_n, v_0)-temporal paths, assuming that the earliest-starting temporal path is a (v_0, v_n)-temporal path that starts at time t. If it is not possible to route such $(k-i)+(\ell-j)$ temporally non-intersecting temporal paths starting at time t, then let $L(i, j, t) = \infty$. Similarly we define $R(i, j, t)$, with the only difference that here the earliest-starting temporal path needs to start at time t from v_n and finishes at v_0. For the sake of completeness, we let $L(i, j, \infty) = R(i, j, \infty) = \infty$ for every $i \leq k$ and every $j \leq \ell$. Furthermore, for every t, every $i \leq k-1$, and every $j \leq \ell - 1$, we let $L(k, j, t) = R(i, \ell, t) = \infty$. Finally we let $L(k, \ell, t) = R(k, \ell, t) = t - 1$. Note that, the input instance I is a *yes*-instance if and only if $\min\{L(0, 0, 1),\ R(0, 0, 1)\} \neq \infty$. Furthermore, note that, for every triple i, j, t, the value $\min\{L(i, j, t), R(i, j, t)\}$ is the earliest arrival time of all temporal paths in the subproblem where, until time $t-1$, exactly i and j temporally non-intersecting temporal (v_0, v_n)- and (v_n, v_0)-paths, respectively, have been routed.

The value $L(i, j, t)$ can be recursively computed as follows. Suppose that, in the optimal solution, $1 \leq p \leq k-i$ temporally non-intersecting (v_0, v_n)-temporal paths are first routed (starting at time t) before the first (v_n, v_0)-temporal path (among the $\ell - j$ ones) is routed. Let t_p be the earliest arrival time of these p paths if they can all be routed; if not, then we set $t_p = \infty$. Then:

$$L(i, j, t) = \min\{R(i+p, j, t_p + 1) \mid 1 \leq p \leq k-i\}. \tag{3.1}$$

The value $R(i, j, t)$ can be computed similarly:

$$R(i, j, t) = \min\{L(i, j + p, t^*_p + 1) \mid 1 \leq p \leq \ell - j\}, \tag{3.2}$$

where (v_n, v_0)-temporal paths are routed.

The values $\{t_p \mid 1 \leq p \leq k - i\}$ can be computed as follows. If $p = 1$, then t_p is the arrival time of the (v_0, v_n)-always-foremost temporal path P_1. To determine t_2, we first compute P_1 and then, for every internal vertex v_x of $\mathcal{P}$, if v_x is visited by P_1 within the time interval $[a_x, b_x]$, then we remove from the edges $\{v_{x-1}, v_x\}, \{v_x, v_{x+1}\}$ of $\mathcal{P}$ all labels $l \leq b_x$. In the resulting temporal line we then compute the always-foremost temporal path P_2, which arrives at v_n at time t_2. By applying this procedure iteratively, we either compute p temporally non-intersecting temporal paths $P_1, P_2, \ldots, P_p$, starting at time t and arriving at time t_p, or we conclude that $t_p = \infty$. The values $\{t^*_p \mid 1 \leq p \leq \ell - j\}$ (for the (v_n, v_0)-temporal paths) can be computed in a symmetric way. All these computations together can be done in linear time.

From the above, it follows that we can decide Temporally Disjoint Paths by checking whether $\min\{L(0, 0, 1), R(0, 0, 1)\}$ is finite or not. In total, there are $2k\ell T$ values $L(i, j, t)$ and $R(i, j, t)$. Observe that, for every pair i, j, we only need to remember the value $L(i, j, t)$ (resp. $R(i, j, t')$) for the smallest value of t (resp. t'). Therefore, we build two memoization matrices M^L and M^R, each of size $(k + 1) \times (\ell + 1)$, such that $M^L(i, j)$ (resp. $M^R(i, j)$) stores the smallest value of t for which we need to compute $L(i, j, t)$ (resp. $R(i, j, t)$). If during the calculation we see that the value of $M^L(i, j)$ is smaller than the current value t of $L(i, j, t)$, we set the value of $L(i, j, t)$ to be infinity and stop with the calculation of this branch.

Similarly, for the recursion tree originating at $R(0, 0, 1)$ we need to build two other matrices N^L and N^R (each of size $(k + 1) \times (\ell + 1)$) for the same purpose, as the recursion tree originated at $R(0, 0, 1)$ is different to the one originated at $L(0, 0, 1)$.

Each of these four $(k+1)\times(\ell+1)$ matrices can be computed by running $O(Tk\ell(k+\ell))$ times the always-foremost temporal path algorithm (in order to compute at each step in linear time $O\left(|V| + \sum_{i=1}^{T}|E_i|\right)$ the values $\{t_p \mid 1 \leq p \leq k - i\}$ and $\{t_p^* \mid 1 \leq p \leq \ell - j\}$, respectively). Once we have built these four matrices, we can iteratively compute the value $L(0,0,1)$ (resp. $R(0,0,1)$) in at most $k\ell$ computations, each of which takes at most $O(k+\ell)$ time (see equations (3.1)-(3.2)). Thus, all computations can be done in $O\left(Tk\ell\,(k+\ell) \cdot \left(|V| + \sum_{i=1}^{T}|E_i|\right)\right)$ time.

This completes the proof for the case of Temporally Disjoint Paths. Finally, it is easy to see that in the problem Temporally Disjoint Walks, where the input temporal graph is a temporal line, in any optimal solution, every temporal walk is a temporal path, as every temporal walk is from v_0 to v_n or from v_n to v_0. Hence, the above algorithm for Temporally Disjoint Paths also solves Temporally Disjoint Walks. □

3.4 The case of few source-sink pairs

In this section, we briefly discuss further results from [86] that were not included in this chapter. More specifically, we present the computational complexity of Temporally Disjoint Paths/Walks for the case when the size of the multiset S of source-sink pairs is small and the underlying graph of the input temporal graph is unrestricted. These findings represent the work which was mostly a product of the collaboration between G. B. Mertzios, H. Molter, R. Niedermeier, and P. Zschoche, where my involvement was not significant.

The first result shows that Temporally Disjoint Paths is NP-hard even for two source-sink pairs [86, Theorem 2]. This is a similar situation as for finding vertex-disjoint paths in directed static graphs, which is also NP-hard for two paths [54]. However, in the temporal setting there is a surprising difference between finding walks and paths that does not have an analogue in the static setting. More specifically, we see that Temporally Disjoint Walks is W[1]-hard for the number $|S|$ of source-sink pairs [86, Theorem 3] and is contained in XP for the same parameter [86, Theorem 4].

3.5 Concluding remarks

In this chapter, we introduced temporally disjoint paths and walks, which models the property that agents moving along the paths never meet, even though they might visit the same vertices. We focused our study on temporal lines and trees. We showed the NP-hardness of Temporally Disjoint Paths and Temporally Disjoint Walks, and provided an FPT algorithm for the number of paths. We left open the question of whether we can obtain a similar result for Temporally Disjoint Walks, which was subsequently answered positively by Kunz et al. [91]. Additionally, we presented a polynomial-time algorithm for the restricted case with source-sink pairs consisting of the endpoints of the path.

We believe that this work can be a starting point for the investigation of many well-motivated variants or generalizations of our problem. One can, for example, consider the case where a predefined set of vertices has to be visited by temporal paths, or a certain "amount" of intersection between paths is acceptable. It is also of interest to investigate our problem for restricted temporal path models such as so-called restless temporal paths or walks [17, 27, 124].

CHAPTER 4

The Complexity of Temporal Vertex Cover in Small-Degree Graphs

This chapter is based on a joint work with Thekla Hamm, George B. Mertzios and Paul G. Spirakis. The preliminary results were presented in the Proceedings of the 36th Conference on Artificial Intelligence (AAAI) [73]. The full paper, containing our detailed results, is accessible as a preprint on ArXiv [74]. As of winter 2023-2024, it is also under the submission process to the Journal of Artificial Intelligence Research (JAIR).

4.1 Introduction

The problems Temporal Vertex Cover (or TVC) and Sliding-Window Temporal Vertex Cover (or Δ-TVC for time-windows of a fixed-length Δ) have been established as natural extensions of the well-known Vertex Cover problem on static graphs [5]. Given a temporal graph $\mathcal{G}$, the aim of TVC is to cover every edge at least once during the lifetime T of $\mathcal{G}$, where an edge can be covered by one of its endpoints, and only at a time-step when it is active. For any $\Delta \in \mathbb{N}$, the aim of the more "pragmatic" problem Δ-TVC is to cover every edge at least once at every Δ consecutive time-steps. In both problems, we want to minimize the number of "vertex appearances" in the resulting cover, where a vertex appearance is a pair (v, t) for some vertex v and $t \in \{1, 2, \ldots, T\}$.

TVC and Δ-TVC naturally generalize the applications of the static problem Vertex Cover to more dynamic inputs, especially in the areas of wireless ad hoc networks, as well as network security and scheduling. In the case of a static graph, the vertex cover can contain trusted vertices which have the ability to monitor/surveil all transmissions [79, 112] or all link failures [82] between any pair of vertices through the edges of the graph. In the temporal setting, it makes sense to monitor the transmissions and to check for link failures within every sliding time window of an appropriate length Δ (which is exactly modelled by Δ-TVC).

It is already known that both TVC and Δ-TVC are NP-hard; for Δ-TVC this is even the case when $\Delta = 2$ and the minimum degree of the underlying graph G is just 3 [5]. One of the most intriguing questions left open (see Problem 1 in [5]) is whether Δ-TVC (or, more generally, Sliding-Window Temporal Vertex Cover) can be solved in polynomial time on always degree at most 2 temporal graphs, that is, on temporal graphs where the maximum degree of the graph at each time-step is at most 2.

Our Contribution. In this work, we present the study of the complexity of TVC and Δ-TVC on sparse graphs. Our main result (see Section 4.3.1) is that, for every $\Delta \geq 2$, Δ-TVC is NP-hard even when G is a path or a cycle. This resolves the first open question (Problem 1) of [5]. In contrast, we show that TVC (see Section 4.3.2) can be solved in polynomial time on temporal lines and cycles. Moreover, for any $\Delta \geq 2$, we provide a *Polynomial-Time Approximation Scheme (PTAS)* for Δ-TVC on temporal lines and cycles (see Section 4.3.2), which also complements our hardness result for temporal lines.

The NP-hardness of Section 4.3.1 signifies that an optimum solution for Δ-TVC is hard to compute, even for $\Delta = 2$ and under severe degree restrictions of the input instance. To counter this hardness for more general temporal graphs than those with underlying paths and cycles as in Section 4.3, in Section 4.4 we give three algorithms for every $\Delta \geq 2$: First we present an exact algorithm for Δ-TVC with exponential running time dependency on the number of edges in the underlying graph (see Section 4.4.1). Using this algorithm we are able to devise, for any $d \geq 3$, a polynomial-time $(d-1)$-approximation (see Section 4.4.2), where d is the maximum vertex degree in any time-step, i. e., in any part of the temporal graph that is active at the same time. This improves the currently best known d-approximation algorithm for Δ-TVC [5] and thus also answers another open question (Problem 2 in [5]). Finally, we present a simple fixed-parameter tractable algorithm with respect to the size of an optimum solution (see Section 4.4.3).

4.2 Preliminaries

Throughout this chapter, we consider temporal graphs whose underlying graphs are finite and whose time-labeling functions only map to finite sets. This implies that there is some $t \in \mathbb{N}$ such that, for every $t' > t$, no edge of G is active at t' in (G, λ). We denote the smallest such t by T, i. e., $T = \max\{t \in \lambda(e) \mid e \in E\}$, and call T the *lifetime* of (G, λ). Unless otherwise specified, n denotes the number of vertices in the underlying graph G, and T denotes the lifetime of the temporal graph $\mathcal{G}$. We refer to each integer $t \in [T]$ as a *time slot* of (G, λ). The *instance* (or *snapshot*) of (G, λ) *at time* t is the static graph $G_t = (V, E_t)$, where $E_t = \{e \in E : t \in \lambda(e)\}$.

Recall, a *temporal line* of length k is a temporal graph $\mathcal{P} = (P, \lambda)$, where the underlying graph P is the path $(v_0, v_1, v_2, \ldots, v_k)$ on $k + 1$ vertices, with edges $e_i = v_{i-1}v_i$ for $i = 1, 2, \ldots, k$. In many places throughout this chapter, we visualize a temporal line as a 2-dimensional array $V(P) \times [T]$, where two vertices $(x, t), (y, t') \in V(P) \times [T]$ are connected in this array if and only if $t = t' \in \lambda(xy)$ and $xy \in E(P)$. For example see Figure 4.1.

For every $t = 1, \ldots, T - \Delta + 1$, let $W_t = [t, t + \Delta - 1]$ be the Δ-time window that starts at time t. For every $v \in V$ and every time slot t, we denote the *appearance of vertex v at time t* by the pair (v, t) and the *edge appearance* (or *time-edge*) of e at time t by (e, t).

A *temporal vertex subset* of (G, λ) is a set of vertex appearances in (G, λ), i.e. a set of the form $S \subseteq \{(v, t) \mid v \in V, t \in [T]\}$. For a temporal vertex subset S and

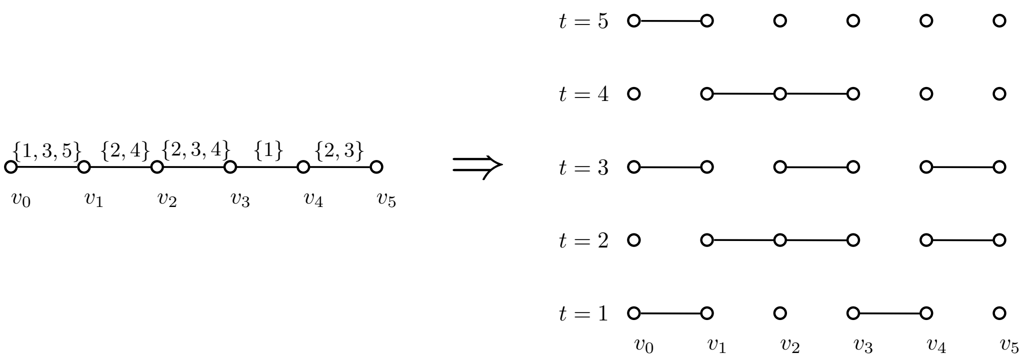


Figure 4.1: An example of visualizing a temporal line graph $\mathcal{G}$ as a 2-dimensional array, in which every edge corresponds to a time-edge of $\mathcal{G}$.

some Δ-time window W_i within the lifetime of (G, λ), we denote by $S[W_i] = \{(v, t) \in S \mid t \in W_i\}$ the subset of all vertex appearances in S in the Δ-time window W_i. For a Δ-time window W_i within the lifetime of a temporal graph (G, λ), we denote by $E[W_i] = \{e \in E \mid \lambda(e) \cap W_i \neq \emptyset\}$ the set of all edges which appear at some time slot within W_i.

A temporal vertex subset $\mathcal{C}$ is a *sliding Δ-time window temporal vertex cover*, or Δ-TVC for short, of a temporal graph (G, λ) if, for every Δ-time window W_i within the lifetime of (G, λ) and for every edge in $E[W_i]$, there is some $(v, t) \in \mathcal{C}[W_i]$ such that $v \in e$, i.e. v is an endpoint of e, and $t \in \lambda(e)$. Here we also say (v, t) *covers* (e, t) in time window W_i.

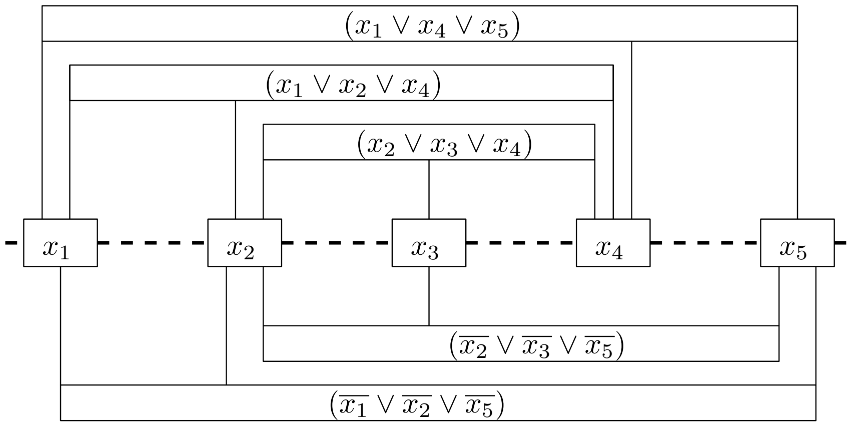
4.3 Paths and cycles

In Section 4.3.1 we provide our main NP-hardness result for Δ-TVC, for any $\Delta \geq 2$, on instances whose underlying graph is a path or a cycle (see Theorem 4.3.8 and Corollary 4.3.9). In Section 4.3.2 we prove that TVC on underlying paths and cycles is polynomially solvable, and we also provide our PTAS for Δ-TVC on underlying paths and cycles, for every $\Delta \geq 2$.

4.3.1 Hardness on temporal lines and cycles

Our NP-hardness reduction of Theorem 4.3.8 is done from the NP-hard problem *planar monotone rectilinear* 3 *satisfiability* (or *planar monotone rectilinear 3SAT*), see [35]. This is a specialization of the classical Boolean 3-satisfiability problem to a planar incidence graph. A Boolean satisfiability formula ϕ in conjunctive normal form (CNF) is called *monotone* if each clause of ϕ consists of only positive or only negative literals. We refer to these clauses as *positive* and *negative* clauses, respectively. By possibly repeating literals, we may assume without loss of generality that every clause contains exactly three (not necessarily different) literals.

In an instance of planar monotone rectilinear 3SAT, each variable and each clause is represented with a horizontal line segment, as follows. The line segments of all variables lie on the same horizontal line on the plane, which we call the *variable-axis*. For every clause $C = (x_i \vee x_j \vee x_k)$ (or $C = (\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$), the line segment of C is connected via straight vertical line segments to the line segments of x_i, x_j and of x_k, such that every two (horizontal or vertical) line segments are pairwise non-intersecting. Furthermore, every line segment of a positive (resp. negative) clause lies above (resp. below) the variable-axis. Finally, by possibly slightly moving the clause line segments higher or lower, we can assume without loss of generality that every clause line segment lies on a different horizontal line on the plane. For an example see Figure 4.2.



$(x_1 \vee x_4 \vee x_5)$
$(x_1 \vee x_2 \vee x_4)$
$(x_2 \vee x_3 \vee x_4)$
x_1
x_2
x_3
x_4
x_5
$(\overline{x_2} \vee \overline{x_3} \vee \overline{x_5})$
$(\overline{x_1} \vee \overline{x_2} \vee \overline{x_5})$

Figure 4.2: An example of an instance of a planar monotone rectilinear 3SAT $\phi = (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_5}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_5})$. For visual purposes, the line segments for variables and for clauses are illustrated here with boxes.

Let ϕ be an arbitrary instance of planar monotone rectilinear 3SAT, where $X = \{x_1, \dots, x_n\}$ is its set of Boolean variables and $\phi(X) = \{C_1, \dots, C_m\}$ is its set of clauses. We construct from ϕ a temporal line $\mathcal{G}_\phi$ and prove (see Lemma 4.3.7) that there exists a truth assignment of X which satisfies $\phi(X)$ if and only if the optimum value of 2-TVC on $\mathcal{G}_\phi$ is at most $f(\mathcal{G}_\phi)$. The exact value of $f(\mathcal{G}_\phi)$ will be defined later.

High-level description

Given a representation (i.e. embedding) R_ϕ of an instance ϕ of planar monotone rectilinear 3SAT, we construct a 2-dimensional array of the temporal line $\mathcal{G}_\phi$, where:

- every variable (horizontal) line segment in R_ϕ is associated with one or more *segment blocks* (to be formally defined below) in $\mathcal{G}_\phi$, and
- every clause (horizontal) line segment in R_ϕ, corresponding to the clause $C = (x_i \vee x_j \vee x_k)$ (resp. $C = (\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$), is associated with a clause gadget in $\mathcal{G}_\phi$, which consists of three edges (one for each of x_i, x_j, x_k), each appearing in 4 consecutive time-steps, together with two paths connecting them in the 2-dimensional array for $\mathcal{G}$ (we call these paths the *clause gadget connectors*, for an illustration see Figure 4.9),
- every vertical line segment in R_ϕ, connecting variable line segments to clause line segments, is associated with an edge of $\mathcal{G}_\phi$ that appears in consecutive time-steps.

The exact description of the variables' and clauses' gadgets is given below; first, we need to precisely define the segment blocks.

Segment blocks are used to represent variables. Every segment block consists of a path of length 7 on vertices $(u_0, u_1, \ldots, u_7)$, where the first and last edges (i. e., u_0u_1 and u_6u_7) appear at 9 consecutive time-steps starting at time t and ending at time $t + 8$, with all other edges appearing only two times, i. e., at times $t + 1$ and $t + 7$. For an example see Figure 4.3.

Time-edges which correspond to the first and last appearances of u_0u_1 and u_6u_7 in a segment block are called *dummy time-edges*, all remaining time-edges form two (bottom and top) horizontal paths, and two (left and right) vertical sequences of time-edges (which we call here *vertical paths*), see Figure 4.4. Using the next

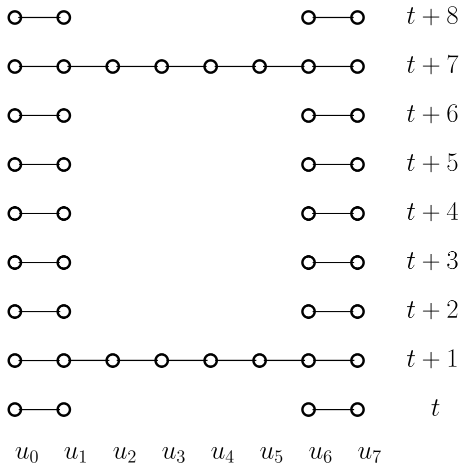


Figure 4.3: Example of a segment block construction.

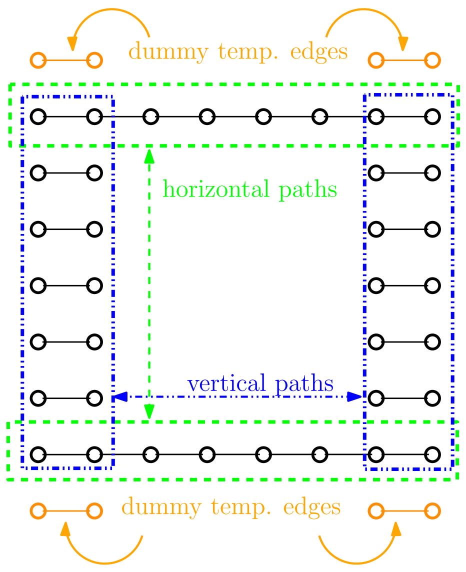


Figure 4.4: An example where dummy time-edges, vertical and horizontal paths are depicted.

technical lemma will allow us to model the two different truth values of each variable x_i (True, resp. False) via two different optimum solutions of 2-TVC on a segment block (namely the "orange and green", resp. "orange and red" temporal vertex covers of the segment block, see Figure 4.5).

Lemma 4.3.1. *There are exactly two different optimum solutions for* 2*-TVC of a segment block, both of size* 15*.*

Proof. Let $\mathcal{C}$ be a 2-TVC of a segment block on vertices $u_0, \ldots, u_7$ that starts at time t and finishes at time $t+8$.

To cover the dummy time-edges in time windows W_{t-1} and W_{t+8} one of their endpoints has to be in $\mathcal{C}$. Now let us start with the covering of the first edge (u_0u_1) at time $t+1$. Since the dummy time-edges are covered, the edge u_0u_1 is covered in the time window W_t but it is not yet covered in the time window W_{t+1}. We have two options, either cover it at time $t+1$ or $t+2$.

- Suppose that we cover the edge u_0u_1 at $t+1$, then the next time-step it has to be covered is $t+3$, the next one $t+5$ and the last one $t+7$. Now that the left vertical path is covered we proceed to cover the bottom and top horizontal paths. The middle 5 edges, from u_1 to u_6, appear only at time-steps $t+1$ and $t+7$. Since we covered the edge u_0u_1 at time $t+1$, we can argue that the optimum solution includes the vertex appearance $(u_1, t+1)$ and therefore the edge u_1u_2 is also covered. Extending this covering optimally to the whole path we need to add every second vertex to $\mathcal{C}$, i. e., vertex appearances $(u_3, t{+}1)$ and $(u_5, t+1)$. Similarly, it holds for the vertex appearances of vertices $u_1, \ldots, u_6$ at time $t+7$. The last thing we need to cover is the right vertical path. Since the edge u_6u_7 is covered at time t, the next time-step we have to cover it is $t+2$, which forces the next cover to be at $t+4$ and the last one at $t+6$.

  In total $\mathcal{C}$ consists of 4 endpoints of the dummy time-edges, 4 vertices of the left and 3 of the right vertical paths, 2 vertices of the bottom and 2 of the top horizontal paths. Altogether we used 11 vertices to cover vertical and horizontal paths and 4 for dummy time-edges. The above described 2-TVC corresponds to the "orange and red" vertex appearances of the odd segment block depicted in Figure 4.5.

  Let us also emphasize that, except for times $t{+}1$ and $t{+}7$, we do not distinguish between the solutions that use a different endpoint to cover the first and last edge. For example, if a solution covers the edge u_0u_1 at time $t+2$ then we do not care which of $(u_0, t+2)$ or $(u_1, t+2)$ is in the TVC.

- Covering the edge u_0u_1 at time $t+2$ produces the 2-TVC that is a mirror version of the previous one on the vertical and horizontal paths. More precisely, in this case the covering consists of 3 vertices of the left and 4 of the right vertical paths and again 2 vertices of the bottom and 2 of the top horizontal paths, together with 4 vertices covering the dummy time-edges.

This 2-TVC corresponds to the “orange and green” vertex appearances of the segment block depicted in Figure 4.5.

If we start with vertex appearances from one solution and add vertex appearances from the other solution then we either create a 2-TVC of bigger size or leave some edges uncovered. Therefore, the optimum temporal vertex cover of any segment block consists of only “orange and red” or “orange and green” vertex appearances. □

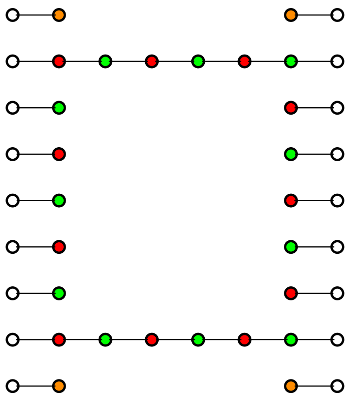


Figure 4.5: An example of two optimum covers of a segment block: (i) with the “orange and green”-colored, or (ii) with the “orange and red”-colored vertex appearances.

For each variable x_i we create multiple copies of segment blocks, and some specific pairs of these segment blocks are connected to each other via the so-called “horizontal bridges”. Two segment blocks, which are connected via a horizontal bridge, start at the same time t but are built on different sets of vertices (i. e., one is to the left of the other in the 2-dimensional array). All the copies have to be created in such a way, that their optimum 2-TVCs depend on each other. In the following, we describe how to connect two different segment blocks (both for the same variable x_i). As we prove below (see Lemma 4.3.2), there are two ways to optimally cover this construction: one using the “orange and green”, and one using the “orange and red” vertex appearances (thus modelling the truth values True and False of variable x_i in our reduction), see Figure 4.6.

Consistency in the horizontal direction. Let $\mathcal{H}$ be a temporal line on vertices $(u_0^1, u_1^1, \ldots, u_7^1, w_1, w_2, w_3, w_4, u_0^2, u_1^2, \ldots, u_7^2)$, with two segment blocks on vertices $(u_0^1, u_1^1, \ldots, u_7^1)$ and $(u_0^2, u_1^2, \ldots, u_7^2)$, respectively, starting at time t and finishing at time $t+8$, and let $P = (u_6^1, u_7^1, w_1, w_2, w_3, w_4, u_0^2, u_1^2)$ be a path of length 7, that appears exactly at times $t+2$ and $t+5$. For an example see Figure 4.6. We refer to this temporal line P as a *horizontal bridge* between the two segment blocks.

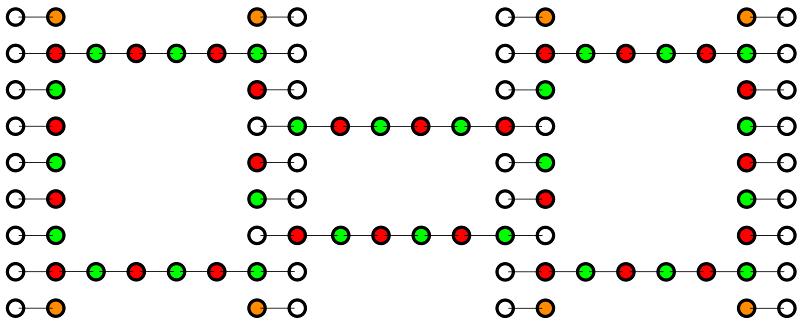


Figure 4.6: An example of connecting two segment blocks in the horizontal direction. The two different ways to optimally cover this construction are (i) with the green-colored and the orange-colored, or (ii) with the red-colored and the orange-colored vertex appearances.

Lemma 4.3.2. *The temporal graph $\mathcal{H}$ has exactly two optimum 2-TVCs, both of size* 34*.*

Proof. Let $\mathcal{H}$ consist of two segment blocks. Suppose that the first segment block, i. e., left one, is covered with the "orange and green" 2-TVC. Then the first edge $u_6^1 u_7^1$ of the path P does not have to be covered at time $t+2$ as it is covered at times $t+1$ and $t+3$. For the other 6 edges of P, there exists a unique optimum 2-TVC, that uses every second vertex at time $t+2$ and is of size 3. Therefore we cover the edge $u_0^2 u_1^2$ at time $t+2$ which enforces the "orange and green" 2-TVC also on the right segment block. Now, the only remaining uncovered edges are four consecutive edges of P, from w_1 to u_0^2 at time $t+5$, which are optimally covered with two vertex appearances. Altogether we used $15+15$ vertex appearances, to cover both segment blocks and $2+2$ to cover P at $t+2, t+5$. □

Lemma 4.3.2 ensures that, in an optimum 2-TVC of the construction of Figure 4.6, either both the left and the right segment blocks contain the "orange and green" vertex appearances, or they both contain the "orange and red" vertex appearances. We can extend the above result to d consecutive copies of the same segment block and get the following result.

Corollary 4.3.3. *The temporal graph corresponding to d copies of a segment block, where two consecutive are connected via horizontal bridges, has exactly two optimum 2-TVCs of size $19d - 4$.*

Proof. The optimum 2-TVC of each segment block consists of 15 vertex appearances. A horizontal bridge is optimally covered using 4 extra vertex appearances. Therefore all together we have $15d + 4(d - 1) = 19d - 4$ vertex appearances in the optimum 2-TVC. □

Variable gadget

From the planar, rectilinear embedding R_ϕ of ϕ, we can easily fix the order of variables. We fix the variables in the order they appear in the variable-axis, starting from the left to the right.

Let d_i be the number of appearances of variable x_i as a literal (i. e., as x_i and $\overline{x_i}$) in ϕ. For every variable x_i we create d_i copies of the segment block, which follow each other on a horizontal line and are connected via *horizontal bridges.* Between variable gadgets of two consecutive variables x_i and x_{i+1} we add 4 vertices (without any additional time-edges). All variable gadgets in $\mathcal{G}_\phi$ start and finish at the same time i. e., they lie on the same horizontal line.

Lemma 4.3.4. *The distance between two rightmost (or two leftmost) edges of any pair of segment blocks in $\mathcal{G}_\phi$ is odd.*

Proof. Let (u_6u_7, t) and (v_6v_7, t) be the rightmost time-edges of two segment blocks $\mathcal{X}$ and $\mathcal{Y}$ in $\mathcal{G}_\phi$. Without loss of generality, we may assume that $\mathcal{X}$ appears before/left of the $\mathcal{Y}$ in $\mathcal{G}_\phi$.

If $\mathcal{X}$ and $\mathcal{Y}$ are consecutive (i. e., right next to each other) then there are 4 vertices between them and two blocks are on the distance 5. Since (v_6v_7, t) is the last (rightmost) time-edge in $\mathcal{Y}$ it is on the distance 6 from the beginning of the block and therefore on the distance 11 from (u_6u_7, t) .

If there are k segment blocks between $\mathcal{X}$ and $\mathcal{Y}$, there are $8k + 4(k + 1)$ vertices between the end of $\mathcal{X}$ and the beginning of the $\mathcal{Y}$. Since (v_6v_7, t) is the last (rightmost) time-edge in $\mathcal{Y}$ there are 6 other vertices before it in the block. Therefore, there are $12k + 10$ vertices between the two edges (u_6u_7, t) and (v_6v_7, t), so they are on the distance $2(6k + 5) + 1$.

Similarly it holds for the leftmost time-edges (u_0u_1, t) and (v_0v_1, t) of two segment blocks $\mathcal{X}$ and $\mathcal{Y}$ in $\mathcal{G}_\phi$. □

Vertical line gadget

A vertical line gadget is used to connect a variable gadget to the appropriate clause gadget. It consists of one edge appearing in $2k$ consecutive time-steps, where k is a positive integer. The value of k for each clause gadget will be specified later; it depends on the embedding R_ϕ of the input formula ϕ. More precisely, let $\mathcal{X}$ be a segment block for the literal x_i (resp. $\overline{x_i}$) on vertices $(u_0, u_1, \ldots, u_7)$, which starts at time t and finishes at time $t' = t + 8$. Then the *vertical line gadget* $\mathcal{V}$ of this segment block consists of just the $2k$ appearances of the edge u_6u_7 from time t' to time $t' + 2k - 1$ (resp. of the edge u_0u_1 from time $t - 2k + 1$ to time t).

Lemma 4.3.5. *Let $\mathcal{V}$ be any vertical line gadget, whose edge appearances are between time t_0 and time $t_0 + 2k - 1$. Then a minimum 2-TVC of $\mathcal{V}$ in the time windows from W_{t_0} to W_{t_0+2k-2} is of size k.*

Proof. To cover an edge in a specific time-step we use just one vertex appearance. Since $\mathcal{V}$ consists of $2k$ consecutive appearances of the same edge u_xu_{x+1}, containing in the 2-TVC an appearance of u_x or u_{x+1} at a time t (where $t_0+1 \leq t \leq t_0+2k-2$) temporally covers the edge u_xu_{x+1} in two time windows, namely W_{t-1} and W_t. Note that there are $2k - 1$ time windows between W_{t_0} to W_{t_0+2k-2}.

Suppose that there is an optimum 2-TVC of $\mathcal{V}$ of size at most $k-1$. This 2-TVC then can cover the edge u_xu_{x+1} in at most $2(k - 1)$ time windows, which leaves at least one time window uncovered. Therefore the size of an optimum 2-TVC is at least k.

Let us now build two minimum 2-TVCs of $\mathcal{V}$, both of size k. We observe the following two options.

- Suppose we cover $u_x u_{x+1}$ at time t_0. Then we can cover it also at times $t_0+2, t_0+4, \ldots, t_0+2k-2$, and thus $u_x u_{x+1}$ is covered in all time windows between W_{t_0} to W_{t_0+2k-2} by using exactly k vertex appearances. This 2-TVC corresponds to the red-colored vertex appearances of the vertical line gadget depicted in Figure 4.7.

- Suppose we cover $u_x u_{x+1}$ at time t_0+1 (instead of time t_0). Then we can cover it also at times $t_0+3, t_0+5, \ldots, t_0+2k-1$, and thus $u_x u_{x+1}$ is covered again in all time windows between W_{t_0} to W_{t_0+2k-2} by using exactly k vertex appearances. This 2-TVC corresponds to the green-colored vertex appearances of the vertical line gadget depicted in Figure 4.7.

□

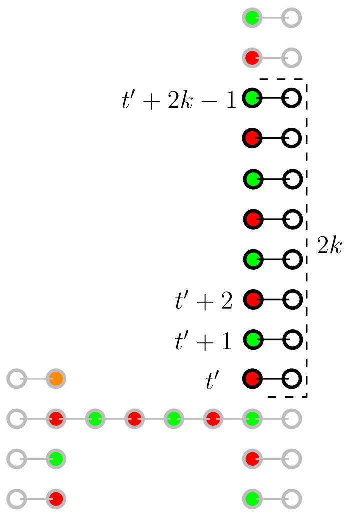


Figure 4.7: An example of two optimum covers of a vertical line gadget: (i) with the green-colored, or (ii) with the red-colored vertex appearances.

Clause gadget

Without loss of the generality, we can assume that in every positive clause $(x_i \vee x_j \vee x_k)$ (resp. negative clause $(\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$) the literals are ordered such that $i \leq j \leq k$. Every clause gadget corresponding to a positive (resp. negative) clause consists of:

- three edges appearing in 4 consecutive time-steps, these edges correspond to the rightmost (leftmost, in the case of a negative clause) edge of a segment block of each variable,

- two paths P_1 and P_2, each of odd length, such that P_1 (resp. P_2) connects the second-to-top (or second-to-bottom, in the case of a negative clause) newly added edges above (below, in the case of a negative clause) the segment blocks of x_i and x_j (resp. of x_j and x_k). We call paths P_1 and P_2 *clause gadget connectors.*

For an example see Figure 4.8.

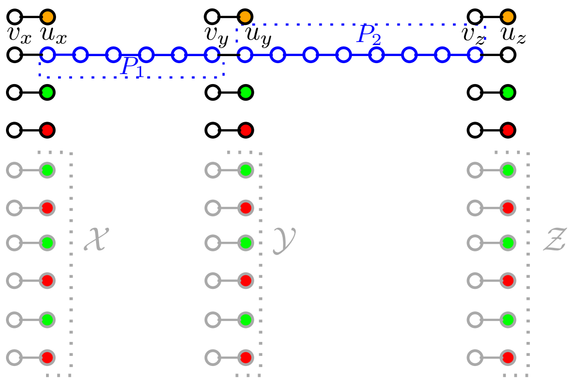


Figure 4.8: An example of a positive clause gadget, depicted with vertical line gadgets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ connected to it.

Lemma 4.3.6. *Let C be a positive clause $(x_i \vee x_j \vee x_k)$ (resp. negative clause $(\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$) and let $\mathcal{C}$ be its corresponding clause gadget in $\mathcal{G}_\phi$. Let the clause gadget connectors P_1 and P_2 have lengths p_1 and p_2, respectively. If we temporally cover at least one of the three vertical line gadgets connecting to $\mathcal{C}$ with the green (resp. red) vertex appearances, then $\mathcal{C}$ can be covered with exactly $7 + \frac{p_1+p_2}{2}$ vertex appearances; otherwise we need at least $8 + \frac{p_1+p_2}{2}$ vertex appearances to temporally cover $\mathcal{C}$.*

Proof. Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ be the vertical line gadgets connecting the segment blocks of variables x_i, x_j, x_k to the clause gadget, respectively and let C be a positive clause. Suppose that $\mathcal{C}$ starts at time t. We denote with v_xu_x, v_yu_y, v_zu_z the underlying edges of $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ respectively. The clause gadget consists of edges v_xu_x, v_yu_y, v_zu_z appearing in consecutive time-steps from t to $t+3$, together with the clause gadget connectors, i. e., a path P_1 of length p_1 from u_x to v_y at time $t+2$ and a path P_2 of length p_2 from u_y to v_z at time $t+2$. Note p_1 and p_2 are odd (see Lemma 4.3.4).

A 2-TVC of $\mathcal{C}$ always covers all three time-edges at time $t+3$, as the time window W_{t+3} needs to be satisfied. If $\mathcal{X}$ is covered with the red vertex appearances, then we need to cover its underlying edge at time t, to satisfy the time window W_{t-1}. Besides that, we also need one extra vertex appearance for the time window W_{t+1}. Without loss of generality, we can cover the edge at time $t+2$. If $\mathcal{X}$ is covered with the green vertex appearances, then the edge $v_x u_x$ is already covered in the time window W_{t-1}, so we cover it at time $t+1$, which satisfies the remaining time windows W_t and W_{t+1}. Similarly it holds for $\mathcal{Y}, \mathcal{Z}$.

We still need to cover the clause gadget connectors P_1 and P_2. Since P_1 and P_2 are paths of odd length appearing at only one time-step, their optimum 2-TVC is of size $(p_1+1)/2$ and $(p_2+1)/2$, respectively. Therefore, if one endpoint of P_1 or P_2 is in the 2-TVC, then to cover them optimally the other endpoint cannot be in the cover.

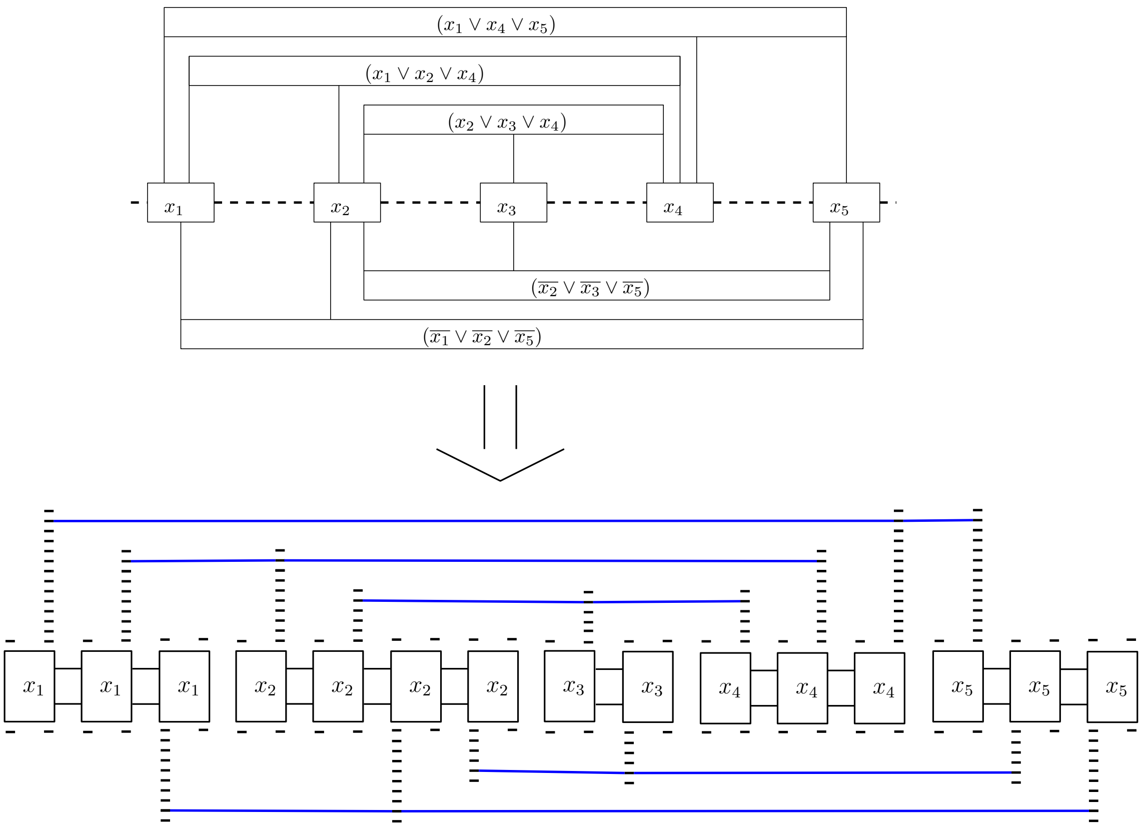
If $\mathcal{Y}$ is covered with the red 2-TVC, then one of the endpoints of the time-edge $(v_y u_y, t+2)$ has to be in the 2-TVC, in the other case, the time-edge is already covered by a vertex appearance in the previous time-step. If $\mathcal{X}$ (resp. $\mathcal{Z}$) is covered with the red 2-TVC, the first (resp. last) time-edge of P_1 (resp. P_2) has to be covered, thus vertex appearance $(u_x, t+2)$ (resp. $(v_z, t+2)$) is in the 2-TVC. Therefore if all $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ are covered with the red 2-TVC we use $(p_1+p_2)/2+2$ vertex appearances to cover P_1 and P_2.

Altogether we need

- one vertex appearance for each edge at time $t+3$,
- one vertex appearance for each edge at time-steps t and $t+1$ and
- either $(p_1+p_2)/2+1$ or $(p_1+p_2)/2+2$ vertex appearances to cover the clause gadget connectors.

We have $8+\frac{p_1+p_2}{2}$ vertex appearances to cover $\mathcal{C}$ if all three $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ are covered with the "orange and red" 2-TVC and $7+\frac{p_1+p_2}{2}$ in any other case.

Similarly, it holds when C is a negative clause. $\square$



$(x_1 \vee x_4 \vee x_5)$
$(x_1 \vee x_2 \vee x_4)$
$(x_2 \vee x_3 \vee x_4)$
x_1
x_2
x_3
x_4
x_5
$(\overline{x_2} \vee \overline{x_3} \vee \overline{x_5})$
$(\overline{x_1} \vee \overline{x_2} \vee \overline{x_5})$
x_1
x_1
x_1
x_2
x_2
x_2
x_2
x_3
x_3
x_4
x_4
x_4
x_5
x_5
x_5

Figure 4.9: An example of the construction of a temporal graph from a planar rectilinear embedding of monotone 3SAT.

Creating a temporal line $\mathcal{G}_\phi$

Before we start with the detailed construction we need to fix the following notation.

Notation. From the planar, rectilinear embedding R_ϕ of ϕ, we can easily fix the order of clauses. We fix the clauses by first fixing all the positive clauses and then all negative ones using the order they appear in the R_ϕ. If C_i, C_j are two positive (resp. negative) clauses and $i < j$, then the clause C_i is above (resp. below) the clause C_j in the R_ϕ. For example in Figure 4.2 we have $C_1 = (x_1 \vee x_4 \vee x_5), C_2 = (x_1 \vee x_2 \vee x_4), C_3 = (x_2 \vee x_3 \vee x_4), C_4 = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_5}), C_5 = (\overline{x_2} \vee \overline{x_3} \vee \overline{x_5})$.

Recall that d_i denotes the number of appearances of the variable x_i as a literal (i. e., either x_i or $\overline{x_i}$) in ϕ. For a clause $C_a = (x_i \vee x_j \vee x_k)$ or $C_a = (\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$ we denote with g_i^a, g_j^a and g_k^a the appearances of the literals x_i, x_j and x_k (resp. $\overline{x_i}, \overline{x_j}$ and $\overline{x_k}$) in C_a, i. e., x_i (resp. $\overline{x_i}$) appears in C_a for the g_i^a-th time, x_j (resp. $\overline{x_j}$) appears g_j^a-th time and x_k (resp. $\overline{x_k}$) appears g_k^a-th time. With m_1 we denote the number of positive and with m_2 the number of negative clauses in ϕ. For every clause C_a, we define also the level ℓ_a in the following way. If C_a is a positive clause then ℓ_a equals the number of clauses between it and the variable-axis increased by 1, i. e., the positive clause C_1, which is furthest away from the variable-axis, is on the level m_1, the clause C_{m_1}, which is the closest to the variable axis, is on level 1. Similarly, it happens for the negative clauses, where the value of the level is negative, i. e., the clause C_{m_1+1} is on the level $-m_2$ and the clause C_m is on the level -1.

Now we are ready to combine all the above gadgets and constructions to describe the construction of the temporal line $\mathcal{G}_\phi$.

Detailed construction. For each variable x_i in ϕ we construct its corresponding variable gadget in $\mathcal{G}_\phi$. All variable gadgets lie on the same horizontal line. We create the clause gadget of a positive $C_a = (x_i \vee x_j \vee x_k)$ or negative clause $C_a = (\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$ such that we connect the g_i^a-th copy of the segment block of x_i to it, together with the g_j^a-th copy of the segment block of x_j and g_k^a-th copy of the segment block of x_k. We do this using vertical line gadgets, that connect the corresponding segment blocks to the level ℓ_a where the clause gadget is starting.

Let us now define our temporal graph $\mathcal{G}_\phi = (G, \lambda)$.

- The underlying graph G is a path on $12\sum_{i=1}^{n} d_i - 4$ vertices.
  - A variable gadget corresponding to the variable x_i consists of d_i copies of the segment block, together with $d_i - 1$ horizontal bridges. We need $d_i 8 + (d_i - 1)4 = 12d_i - 4$ vertices for one variable and $\sum_{i=1}^{n}(12d_i - 4)$ for all of them.
  - Between variable gadgets of two consecutive variables x_i, x_{i+1} there are 4 vertices. Therefore we use extra $4(n-1)$ vertices, for this construction.
- The lifetime T of $\mathcal{G}_\phi$ is $4(m+4)$.
  - There are m clauses and each of them lies on its own level. Therefore we need $4m$ time-steps for all clause gadgets.
  - There is a level 0 with all the variable gadgets, of height 9. We add one extra time-step before we define the start of levels 1 and -1. This

ensures that the vertical line gadget, connecting the variable gadget to the corresponding clause gadget, is of even height.

– The first time-edge appears at time 5 instead of 1, which adds 4 extra time-steps to the lifetime of $\mathcal{G}_\phi$ and does not interfere with our construction. Similarly, after the appearance of the last time-edge, there is one extra time-step, where no edge appears. We add these time-steps to ensure that the dummy time-edges must be covered in the time-step they appear and that the lifetime of the graph $\mathcal{G}_\phi$ is divisible by 4.

– All together we have $4m + 11 + 4 + 1$ time-steps.

Therefore $\mathcal{G}_\Phi$ is a temporal graph of lifetime $T = 4(m + 4)$ with the underlying path on $12 \sum_{i=1}^{n} d_i - 4$ vertices. The appearances of each edge arise from the structure of the planar monotone rectilinear 3SAT.

Size of the optimum 2-TVC of $\mathcal{G}_\phi$

Using the notation introduced above we determine the size of the optimum 2-TVC of $\mathcal{G}_\phi$, under the assumption that the input 3SAT formula ϕ is satisfiable. We do this in two steps, first determining the size of the optimum 2-TVC for each variable gadget and then determining the optimum 2-TVC of all vertical line gadgets together with the clause gadgets.

1. Optimum 2-TVC covering all variable gadgets is of size $19 \sum_{i=1}^{n} d_i - 4n$.

   - Since a variable x_i appears in d_i clauses as a positive or negative literal, we construct d_i connected copies of the segment block. By Corollary 4.3.3 this construction has exactly two optimum 2-TVCs, each of size $19d_i - 4$.
   - Using one of the four dummy time-edges of a segment block, denote it (e, t), we connect the corresponding variable gadget to the clause gadget. Therefore we need to cover just 3 dummy time-edges. Since there is one extra time-step before the first clause gadget level, the edge e appears also at time $t + 1$ in the case of a positive clause and $t - 1$ in the case of the negative one. To optimally cover it we then use one vertex appearance.

- There are n variables, so all together we need $\sum_{i=1}^{n}(19d_i - 4)$ vertex appearances in 2-TVC to optimally cover all of them.

2. Optimum 2-TVC covering all vertical line gadgets and clause gadgets is of size

$$\sum_{\substack{C_a=(x_i\vee x_j\vee x_k)\text{ or}\\ C_a=(\overline{x_i}\vee\overline{x_j}\vee\overline{x_k})}} \left(6|\ell_a| + 6\sum_{b=1}^{k-i-1} d_{i+b} + 6(d_i - g_i^a + g_k^a)\right) - 2m. \tag{4.1}$$

Let $C_a = (x_i \vee x_j \vee x_k)$ (resp $C_a = (\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$).

- Vertical line gadgets connect variable gadgets of x_i, x_j and x_k with the clause gadget of C_a. Since the clause gadget is on level ℓ_a the corresponding vertical line gadgets are of height $4(|\ell_a| - 1)$. By Lemma 4.3.5 and the fact that each clause requires exactly three vertical line gadgets, we need $6(|\ell_a| - 1)$ vertex appearances to cover them optimally.
- Using Lemma 4.3.6 we know that an optimum 2-TVC covering the clause gadget corresponding to C_a requires $7+\frac{p_1+p_2}{2}$ vertex appearances where p_1 and p_2 are the lengths of the clause gadget connectors. Let us determine the exact value of $p_1 + p_2$.
  - There are $k - i - 1$ variable gadgets between x_i and x_k, namely $x_{i+1}, x_{i+2}, \dots, x_{k-1}$. Variable gadget corresponding to any variable x_b is of length $12d_b - 4$. There are 4 vertices between two consecutive variable gadgets. Therefore there are $\sum_{d=1}^{k-i-1}(d_{i+d}12 - 4) + (k - i)4 = 12\sum_{d=1}^{k-i-1} d_{i+d} - 4$ vertices between the last vertex of x_i variable gadget and the first vertex of x_k variable gadget.
  - From the end of g_i^a-th segment block of x_i to the end of the variable gadget there are $d_i - g_i^a$ copies of the segment block together with the $d_i - g_i^a$ horizontal bridges connecting them. Similarly, it holds for the case of x_k, there are $g_k^a - 1$ segment blocks and horizontal bridges. Altogether we have $(d_i - g_i^a + g_k^a - 1)8 + (d_i - g_i^a + g_k^a - 1)4 = 12(d_i - g_i^a + g_k^a - 1)$ vertices.
  - By the construction, paths p_1 starts in the last vertex of the g_i^a-th

segment block of the variable gadget of x_i, and p_2 finishes in the first vertex of the g_k^a-th segment block of the variable gadget of x_k. Therefore we add 2 to the value of $p_1 + p_2$.

– The value of $p_1 + p_2$ is

$$12 \sum_{d=1}^{k-i-1} d_{i+d} + 4 + 12(d_i - g_i^a + g_k^a - 1) + 2 = 12 \sum_{d=1}^{k-i-1} d_{i+d} + 12(d_i - g_i^a + g_k^a) - 6.$$

Since the optimum 2-TVC of the clause C_a requires $7 + \frac{p_1+p_2}{2}$ vertex appearances, it is of size

$$6 \sum_{d=1}^{k-i-1} d_{i+d} + 6(d_i - g_i^a + g_k^a) + 4.$$

Altogether, the optimum 2-TVC of the clause C_a with the corresponding vertical line gadgets is of size

$$6|\ell_a| + 6 \sum_{d=1}^{k-i-1} d_{i+d} + 6(d_i - g_i^a + g_k^a) - 2.$$

Extending the above equation to all clauses and variables we get that the optimum 2-TVC of $\mathcal{G}_\phi$ is of size

$$\begin{aligned} s =& 19 \sum_{i=1}^{n} d_i + \sum_{\substack{C_a = (x_i \vee x_j \vee x_k) \text{ or} \\ C_a = (\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})}} \left(6|\ell_a| + 6 \sum_{b=1}^{k-i-1} d_{i+b} + 6(d_i - g_i^a + g_k^a) \right) \\ & - 2m - 4n. \end{aligned} \tag{4.2}$$

We are now ready to prove our main technical result of this section.

Lemma 4.3.7. *There exists a truth assignment τ of the variables in X which satisfies $\phi(X)$ if and only if there exists a feasible solution to 2-TVC on $\mathcal{G}_\phi$ which is of the size s as in Equation (4.2).*

Proof. Each segment block corresponding to a variable x has exactly two optimum 2-TVCs (the "orange and green" and "orange and red" one, depicted in Figure 4.5). We set the "orange and green" solution to the True value of x and the "orange and red" one to False.

($\Rightarrow$): From τ we start building a 2-TVC of $\mathcal{G}_\phi$ by first covering all variable gadgets with a 2-TVC of "orange and green" (resp. "orange and red") vertex appearances if the corresponding variable is True (resp. False). Using Lemma 4.3.2 we know that the optimum 2-TVC of every variable gadget uses either all "orange and green" or all "orange and red" vertex appearances for every segment block and horizontal bridges connecting them. Next, we extend the 2-TVC from variable gadgets to vertical line gadgets, i. e., if the variable gadget is covered with the "orange and green" (resp. "orange and red") vertex appearances then the vertical line gadget connecting it to the clause gadget uses the green (resp. red) vertex appearances. In the end, we have to cover also clause gadgets. Using Lemma 4.3.6 we know that we can cover each clause gadget optimally if and only if at least one vertical line gadget connecting to it is covered with the green vertex appearances, in the case of a positive clause and red vertex appearances in the case of the negative one. More precisely, a clause gadget $\mathcal{C}$ corresponding to the clause C is covered with the minimum number of vertex appearances whenever C is satisfied. Since τ is a truth assignment satisfying ϕ, all clauses are satisfied and hence covered optimally. Therefore the size of the 2-TVC of $\mathcal{G}_\phi$ is the same as in Equation (4.2).

($\Leftarrow$): Suppose now that $\mathcal{C}$ is a minimum 2-TVC of the temporal graph $\mathcal{G}_\phi$ of size s. Using Lemma 4.3.2 and Corollary 4.3.3 we know that the optimum 2-TVC of each variable gadget consists of either all "orange and green" or all "orange and red" vertex appearances. If this does not hold then $\mathcal{C}$ is not of minimum size on variable gadgets. Similarly, Lemma 4.3.6 assures that the number of vertex appearances used in the 2-TVC of any clause gadget is minimum if and only if there is at least one vertical line gadget covered with green vertex appearances in the case of a positive clause and red vertex appearances in the case of a negative clause. If this does not hold then $\mathcal{C}$ is not of minimum size on clause gadgets. Lastly, if a vertical line gadget $\mathcal{V}$ connects a variable gadget covered with "orange and green" (resp. "orange and red") vertex appearances to a clause gadget using "orange and red" (resp. "orange and green") vertex appearances, then the 2-TVC of $\mathcal{V}$ is not minimum (see Lemma 4.3.5) and $\mathcal{C}$ cannot be of size s.

To obtain a satisfying truth assignment τ of ϕ we check for each variable x_i which vertex appearances are used in the 2-TVC of its corresponding variable gadget. If $\mathcal{C}$ covers the variable gadget of x_i with all "orange and green" vertex appearances we set x_i to TRUE else we set it to FALSE. □

Our main result of this section follows now directly from Lemma 4.3.7 and the fact that the planar monotone rectilinear 3SAT is NP-hard [35].

As, for every $\Delta \geq 2$, there is a known polynomial-time reduction from Δ-TVC to $(\Delta + 1)$-TVC [5], we obtain the following.

Theorem 4.3.8. *For every $\Delta \geq 2$, Δ-TVC on instances on an underlying path is NP-hard.*

With a slight modification to the $\mathcal{G}_\phi$ we can create the temporal cycle from R_ϕ and therefore the following holds.

Corollary 4.3.9. *For every $\Delta \geq 2$, Δ-TVC on instances on an underlying cycle is NP-hard.*

Proof. We follow the same procedure as above where we add one extra vertex w, to the underlying graph P of $\mathcal{G}_\phi$. We add also two time-edges connecting the first and the last vertex of the temporal line graph $\mathcal{G}_\phi$ at time 1. This increases the size of the 2-TVC by 1 (as we need to include the vertex appearance $(w, 1)$) and it transforms the underlying path P into a cycle. □

4.3.2 Algorithmic results

To complement the hardness presented in Section 4.3.1, we present two polynomial-time algorithms. Firstly, a dynamic program for solving TVC on instances with underlying paths and cycles, which shows that the hardness is inherently linked to the sliding time windows. Secondly, we give a PTAS for Δ-TVC on instances with underlying paths. This approximation scheme can be obtained using a powerful general purpose result commonly used for approximating geometric problems [105].

Theorem 4.3.10. *TVC can be solved in polynomial time on instances with a path/cycle as their underlying graph.*

Proof. Let us start the proof with the case when the underlying graph G is a path $(v_0, v_1, v_2, \ldots, v_n)$ on $n+1$ vertices. Our algorithm mimics the greedy algorithm for computing a minimum vertex cover in a (static) path graph, which just sweeps the path from left to right and takes every second vertex. For every $i = 1, 2, \ldots, n$ we denote by e_i the edge $v_{i-1}v_i$.

To compute the optimum temporal vertex cover $\mathcal{C}$ of $\mathcal{G}$ we need to determine for each edge which vertex appearance covers it. We first consider the edge v_0v_1. Without loss of generality, we can cover this edge in $\mathcal{C}$ by a vertex appearance in the set $\{(v_1, t) : v_0v_1 \in E_t\}$, as no appearance of vertex v_0 can cover any other edge apart from v_0v_1. We check whether the set $\{t : v_0v_1 \in E_t\} \cap \{t' : v_1v_2 \in E_{t'}\}$ is empty or not; that is, we check whether v_0v_1 appears together with v_1v_2 in any time-step. If this set is empty, then we add to $\mathcal{C}$ an arbitrary vertex appearance (v_1, t), where $(e_1, t) \in E_t$. This choice is clearly optimum, as we need to cover v_0v_1 with at least one vertex appearance of v_1, while any of the vertex appearances of v_1 can not cover any other edge of the graph. Otherwise, let $t \in \{t : v_0v_1 \in E_t\} \cap \{t' : v_1v_2 \in E_{t'}\}$ be an arbitrary time-step in which both v_0v_1 and v_1v_2 appear. Then we add (v_1, t) to $\mathcal{C}$. This choice is again optimum, as in this case the vertex appearance (v_1, t) covers both edges v_0v_1 and v_1v_2. After adding one vertex appearance to $\mathcal{C}$, we ignore the edges that have been covered so far by $\mathcal{C}$, and we proceed recursively with the remaining part of $\mathcal{G}$, until all edges are covered. Each iteration of the above algorithm can be performed in $O(T)$ time, and we have at most $O(n)$ iterations. Therefore the running time of the whole algorithm is $O(Tn)$.

We can similarly deal with the case where the underlying graph G is a cycle on the vertices $v_0, v_1, v_2, \ldots, v_n$. Here we just need to observe that, without loss of generality, the optimum solution contains either a vertex appearance of v_0 or a vertex appearance of v_1 (but not both). Therefore, to solve TVC on $\mathcal{G}$, we proceed as follows. First we create an auxiliary cycle G' on the vertices $v'_0, v_0, v_1, v_2, \ldots, v_n$ (in this order), where v'_0 is a new dummy vertex. Then we create the temporal graph $\mathcal{G}' = (G', \lambda')$, where $\lambda'(v_iv_{i+1}) = \lambda(v_iv_{i+1})$ for every $i = 1, 2, \ldots, n-1$, $\lambda'(v_nv'_0) = \lambda(v_nv_0)$, and $\lambda'(v'_0v_0) = \lambda'(v_0v_1) = \lambda(v_0v_1)$. Next we compute the temporal subgraphs $\mathcal{G}'_1$ and $\mathcal{G}'_2$ of $\mathcal{G}'$, where the underlying graph of $\mathcal{G}'_1$ (resp. of $\mathcal{G}'_2$) is the path $(v_0, v_1, \ldots, v_n, v'_0)$ (resp. the path $(v_1, v_2, \ldots, v_n, v'_0, v_0)$). Every edge of $\mathcal{G}'_1$ and of $\mathcal{G}'_2$ has the same time labels as the corresponding edge in $\mathcal{G}$. Now we optimally solve TVC on $\mathcal{G}'_1$ and on $\mathcal{G}'_2$, and we return the smaller of the two solutions, as an optimum solution of TVC on $\mathcal{G}$ (by replacing any vertex appearance (v'_0, t) by (v_0, t)). The running time is again $O(Tn)$. □

Next, we turn to approximating Δ-TVC on underlying paths. The Geometric Hitting Set problem takes as an instance a set of geometric objects, e.g. shapes in the plane, and a set of points. The task is to find the smallest set of points, that hit all of the objects. In the paper by Mustafa and Ray [105] the authors present a PTAS for the problem, when the geometrical objects are r-admissible set regions. We transform an arbitrary temporal line to the setting of the geometric hitting set. As a result, we obtain a PTAS for the Δ-TVC problem.

Definition 4.3.11. *Let $R = (P, D)$ be a range space that consists of points P in $\mathbb{R}^2$ and a set D of regions containing points of P. The* minimum geometric hitting set *problem asks for the smallest set $S \subseteq P$, such that each region in D contains at least one point from S.*

Definition 4.3.12. *Let D be the set of regions in the plane, where any region $s \in D$ is bounded by a closed Jordan curve* [1]*. We say, that D is an* r-admissible *set of regions, if Jordan curves bounding any two regions $s_1, s_2 \in D$ cross $l \leq r$ times and both $s_1 \setminus s_2$ and $s_2 \setminus s_1$ are connected regions*[2]*.*

Theorem 4.3.13 ([105]). *Let $R = (P, D)$ be a range space consisting of a set P of n points in $\mathbb{R}^2$ and a set D of m r-admissible regions. There exists a $(1 + \varepsilon)$-approximation algorithm for the minimum geometric hitting set problem that is performed in $O(mn^{O(\varepsilon^{-2})})$ time.*

Theorem 4.3.14. *For every $\varepsilon > 0$, there exists an $(1 + \varepsilon)$-approximation algorithm for Δ-TVC on instances with a path/cycle as their underlying graph, which runs in*

[1]A Jordan curve is a non-self-intersecting continuous loop.

[2]The space X is said to be path-connected if there is a path joining any two points in X. Every path-connected space is also connected.

time

$$O\left(n(T-\Delta+1)\cdot(T(n+1))^{O(\varepsilon^{-2})}\right)=O\left((T(n+1))^{O(\varepsilon^{-2})}\right),$$

i. e., the problem admits a PTAS.

Proof. Let $\mathcal{G}=(G,\lambda)$ be a temporal line, on vertices $\{v_1,v_2,\dots,v_n\}$, with lifetime T. We first have to create the range space $R=(P,D)$, where $P\subseteq V\times[T]$ is a set of vertex appearances and D is a set of 2-admissible regions.

Set P of time vertices consist of vertex appearances (v_i,t) for which $t\in\lambda(e_i)\cup\lambda(e_{i+1})$. Intuitively, if edges incident to v do not appear at time t, then (v,t) is not in P. Set D of 2-admissible regions consists of rectangles of 2 different sizes. For every edge e_i that appears in the time window W_t we create one rectangle R_i^t, that includes all vertex appearances incident to e_i in W_t. For example, if edge e_i appears in the time window W_t at times t_1 and t_2, then the corresponding rectangle R_i^t contains vertex appearances $(v_{i-1},t_1),(v_i,t_1),(v_{i-1},t_2)$ and (v_i,t_2).

It is not hard to see that $|P|\leq|V|\cdot T=(n+1)T$ and $|D|\leq|E|(T-\Delta+1)=n(T-\Delta+1)$.

Formal construction.

Since D will be defined to be a set of 2-admissible regions, the boundary of any two rectangles we construct should intersect at most 2 times. For this purpose, we use rectangles, of two different sizes. Let us denote with A the rectangle of size $a_1\times a_2$ and with B be the rectangle of size $b_1\times b_2$, where $a_1>b_1>b_2>a_2$.

As we said, for every edge e_i that appears in the time window W_t we construct exactly one rectangle. These are the rules we use to correctly construct them.

1. For a fixed time window W_t we construct the rectangles in such a way, that they intersect only in the case when their corresponding edges e_i,e_j share the same endpoint in the underlying graph G. Since G is a path, the intersection happens only in the case when $j=i+1$. We can observe also, that there are no three (or more) edges sharing the same endpoint and therefore no three rectangles intersect.

We require also, that the rectangles corresponding to a pair of adjacent edges are not the same, i. e., they alternate between form A and B. For an example see Figure 4.10.

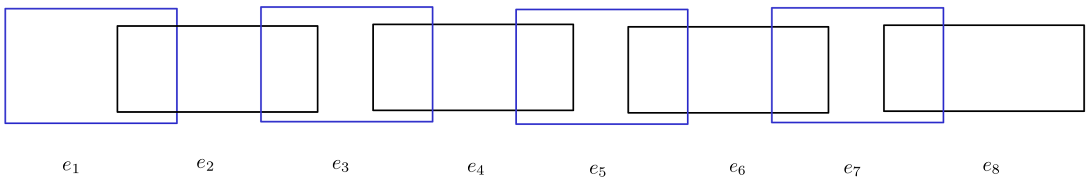
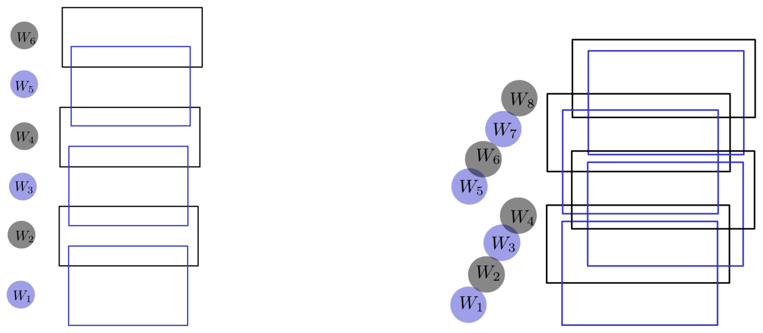


Figure 4.10: Regions in a fixed time window.

2. For any edge e_i, rectangles corresponding to two consecutive time windows W_t, W_{t+1} are not the same, i. e., they alternate between the form A and B. For an example see Figure 4.11a.



(a) Alternating regions for an edge. (b) Regions for an edge with shift, when $\Delta = 4$. Each region W_i starts at same horizontal position as W_{i-4}.

Figure 4.11: Creating regions for one edge.

When the time windows are of size Δ, there are at most Δ rectangles intersecting at every time-step t. This holds, because if the edge e_i appears at time t it is a part of the time windows $W_{t-\Delta+1}, W_{t-\Delta+2}, \ldots, W_t$.

Since the constructed rectangles are of two sizes, if $\Delta \geq 3$ we create intersections with an infinite number of points between the boundary of some rectangles, if we just "stack" the rectangles upon each other. Therefore, we need to shift (in the horizontal direction) rectangles of the same form in one Δ time window. Since the time window W_t never intersects with $W_{t+\Delta}$, we can shift the time windows $W_{t+1}, \ldots, W_{t+\Delta-1}$ and fix the $W_{t+\Delta}$ at the same horizontal position as W_t . For an example see Figure 4.11b.

Combining both of the above rules we get a grid of rectangles. Moving along the x axis corresponds to moving through the edges of the path and moving along the y axis corresponds to moving through the time-steps.

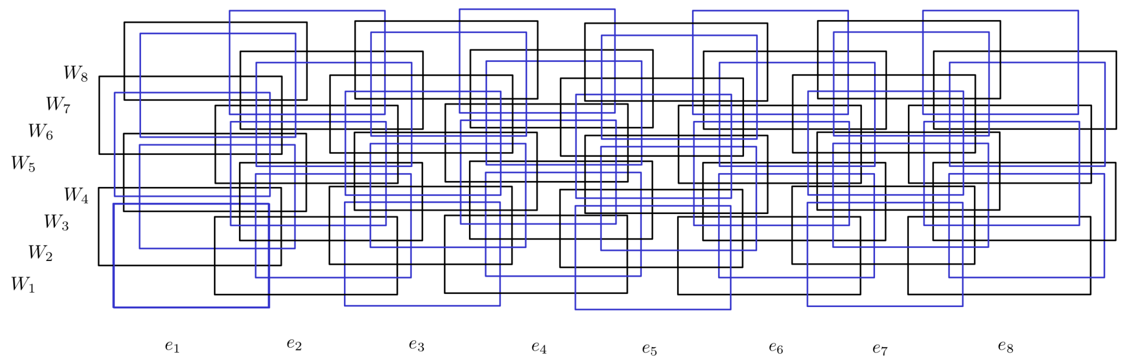


Figure 4.12: Regions for edges $e_1, \ldots, e_8$ in the time windows $W_1, \ldots, W_8$, in the case of 4-TVC.

By Figure 4.10 and Item 2 of the construction above, rectangles alternate between the form A and B in both dimensions. See figure Figure 4.12 for example. If an edge e_i does not appear in the time window W_t, then we do not construct the corresponding rectangle R_i^t. The absence of a rectangle from a grid does not change the pattern of other rectangles. To determine of what form a rectangle corresponding to edge e_i at time-window W_t is, we use the following condition:

$$R_i^t = \begin{cases} A & \text{if } i + t \equiv 0 \pmod 2, \\ B & \text{else.} \end{cases}$$

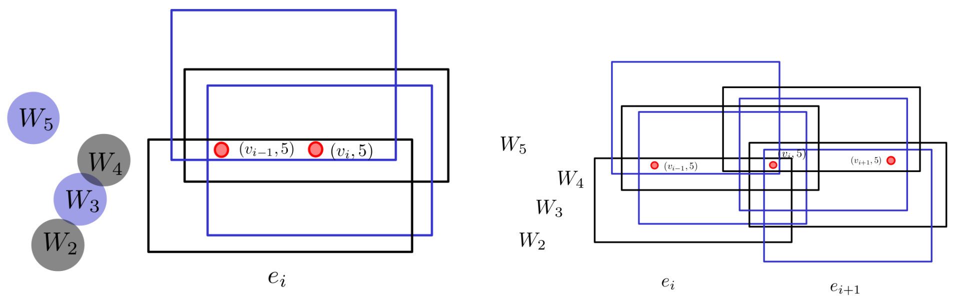
Now we define where the points are placed. We use the following conditions.

a. If an edge $e_i = v_{i-1}v_i$ appears at time t we add vertex appearances $(v_{i-1}, t), (v_i, t)$ to all of the rectangles $R_i^{t'}$, where $t - \Delta + 1 \leq t' \leq t$, if they exist.

   Equivalently, we add the vertex appearances $(v_{i-1}, t), (v_i, t)$ in the intersection of the rectangles corresponding to the edge e_i in the time windows $W_{t-\Delta+1}, \ldots, W_t$. For an example see Figure 4.13a.

b. If an edge e_i does not appear at time t, then the time vertices $(v_{i-1}, t), (v_i, t)$ are not included in the rectangles $R_i^{t'}$ $(t - \Delta + 1 \leq t' \leq t)$, if they exist.

c. If two adjacent edges e_i, e_{i+1} appear at the same time t, we add to the intersection of the rectangles $R_i^{t'}, R_{i+1}^{t'}$ the vertex (v_i, t), where $t - \Delta + 1 \leq t' \leq t$. For an example see Figure 4.13b.



(a) Edge e_i appearing at time 5 and the corresponding placement of vertex appearances to the rectangles, when $\Delta = 4$.

(b) Edges e_i and e_{i+1} appearing at time 5 and the corresponding vertex appearances placement in the rectangles, when $\Delta = 4$.

Figure 4.13: Example of placement of the vertices into the rectangles when $\Delta = 4$.

It is straightforward to verify that finding the minimum hitting set of the range space is equivalent to finding the minimum Δ-TVC for $\mathcal{G}$. On the constructed range space we use the local search algorithm from [105] which proves our result. Note, that we assume that if a region does not admit any points, then it is trivially satisfied. □

4.4 Algorithms for bounded degree temporal graphs

In this section, we extend our focus from temporal graphs with underlying paths or cycles to instances of Δ-TVC with more general degree restrictions. In particular, we present an algorithm that solves Δ-TVC exactly, in time that is exponential in the number of edges of the underlying graph. We then use this algorithm to give a $(d-1)$-factor approximation algorithm (where d is the maximum vertex degree in any time-step) and finally give an FPT-algorithm parameterized by the size of a solution.

Before we present our algorithms let us define the generalized notion of (sub)instances of our Δ-TVC problem. We use this notion in the formulation of the exact exponential time algorithm and in the approximation algorithm.

Definition 4.4.1 (Partial Δ-TVC)**.** *Let (G, λ) be a temporal graph. An instance of* Partial *Δ-TVC is given by (G, λ, ℓ, h), where $\ell : E(G) \to [T]$ and $h : E(G) \to [T]$ map each edge to the starting time of its* lowest uncovered time window *and* highest uncovered time window *respectively. The task is to find a temporal vertex subset $\mathcal{C}$ of minimum size, such that for every edge e and every time window W_i with $\ell(e) \leq i \leq h(e)$ if $e \in E[W_i]$ there is some $(v, t) \in \mathcal{C}[W_i]$ where $v \in e$, $t \in \lambda(e)$.*

Intuitively, Partial Δ-TVC of (G, λ, ℓ, h) is a Δ-TVC of (G, λ), where each edge $e \in E(G)$ has to be covered only in time windows $W_{l(e)}, W_{l(e)+1}, \ldots, W_{h(e)}$. Obviously, if $\ell(e) = 1$ and $h(e) = T - \Delta + 1$ for all $e \in E(G)$ then a solution of Partial Δ-TVC on (G, λ, ℓ, h) is also a valid solution of Δ-TVC on (G, λ).

4.4.1 Exact algorithm

In the following we denote by d_G the degree of the underlying graph G of the considered instances (G, λ, ℓ, h) of Partial Δ-TVC. We provide a dynamic programming algorithm with running time $O(T\Delta^{O(|E(G)|)})$, as the next theorem states.

Theorem 4.4.2. *For every $\Delta \geq 2$, a solution to* Partial *Δ-TVC can be computed in $O(Tc^{O(|E(G)|)})$ time, where $c = \min\{2^{d_G}, \Delta\}$ and T is the lifetime of the temporal graph in the instance.*

Intuitively, our algorithm scans the temporal graph Δ time-steps at a time and for each edge, e_i calculates the optimal vertex appearance that covers it. Since a time-edge (e_i, t) can appear at each time with a different combination of time-edges incident to it (we call this a *configuration* of edges), it is not always the best to simply cover the last appearance of e_i in the observed time window. To overcome this property we show that it is in fact enough to consider the latest appearance of each configuration of e_i, which drastically reduces the search space of our algorithm. Let us now proceed with the detailed proof of Theorem 4.4.2.

Proof. Let (G, λ, ℓ, h) be an instance of PARTIAL Δ-TVC. We denote with m the number of edges in the underlying graph G and with T the lifetime of the input instance, more precisely $T \in W_t$, where $t = \max_{e \in E(G)}(h(e))$. For (G, λ, ℓ, h) we define a dynamic programming table f, that is indexed by tuples $(t, x_1, \ldots, x_m)$, where $t \in [T - \Delta + 1]$ and $x_i \in [0, \Delta]$ for all $i \in [m]$. We store in $f(t, x_1, \ldots, x_m)$ the size of a minimum PARTIAL Δ-TVC of a sub-instance (G, λ, ℓ', h) with $\ell'(e_i) = t + x_i$, i. e., we modify the lowest uncovered time window of each edge. We call such a minimum solution a *witness* for $f(t, x_1, \ldots, x_m)$. Whenever no solution exists we keep $f(t, x_1, \ldots, x_m)$ empty.

The following observation is an immediate consequence of the definition and the fact that $t + x_i = (t + 1) + (x_i - 1)$.

Observation 4.4.3. *If for all i, $x_i \geq 1$ then:*

$$f(t, x_1, x_2, \ldots, x_m) = f(t + 1, x_1 - 1, x_2 - 1, \ldots, x_m - 1).$$

We introduce the notion of *configurations* of edges, that we use to recursively fill the dynamic programming table. Let $v, w \in V(G)$ be the endpoints of edge e_i. An edge configuration of time-edge (e_i, t) at endpoint v (resp. w) consist of all time-edges that are incident to (v, t) (resp. (w, t)). Formally, $\gamma(e_i, t)_v = \{e \in E(G) \mid v \in e_i \cap e, t \in \lambda(e)\}$ is a configuration of edges incident to time-edge (e_i, t) at endpoint v. With $\gamma(e_i)_v = \{\gamma(e_i, t)_v \mid t \in [T], t \in \lambda(e_i)\}$ we denote the *set of all configurations* of edge $e_i \in (G, \lambda)$ at endpoint v. Note, since each vertex in G is of degree at most d_G it follows that e_i is incident to at most $2(d_G - 1)$ edges in G ($d_G - 1$ at each endpoint) and therefore appears in at most 2^{d_G} different edge configurations, i. e. $|\gamma(e_i)_{v_i}| + |\gamma(e_i)_{w_i}| \leq 2^{d_G}$ for all $v_i w_i = e_i \in E(G)$. The applicability of edge configurations is presented by the following claim.

Claim 4.4.4. *Suppose that the edge $vw = e_i$ is optimally (temporally) covered in every time window up to W_t. Then, to determine a temporal vertex cover of e_i in the time window W_t, that is a part of an optimum temporal vertex cover, it is enough to check only the last appearance of every edge configuration of e_i in W_t.*

Proof. Suppose that the configuration of e_i at time t_1 is the same as the configuration of e_i at time t_2 (i.e. $\gamma(e_i,t_1)_v = \gamma(e_i,t_2)_v$ and $\gamma(e_i,t_1)_w = \gamma(e_i,t_2)_w$), where $t \leq t_1 < t_2 \leq t+\Delta-1$. Let $\mathcal{C}$ be an optimum temporal vertex cover of $\mathcal{G}$ that covers edge e_i at time t_1 with the vertex appearance (v,t_1). Now let $\mathcal{C}' = \{\mathcal{C} \setminus (v,t_1)\} \cup (v,t_2)$. Since $\mathcal{C}'$ and $\mathcal{C}$ differ only in the time window W_t all edges in configuration $\gamma(e_i,t_1)_v$ remain optimally (temporally) covered in all time windows up to W_t. Since (v,t_2) temporally covers the same edges as (v,t_1) in the time window W_t, it follows that $\mathcal{C}'$ is also a Δ-TVC. Clearly $\mathcal{C}'$ is not bigger than $\mathcal{C}$. Therefore, $\mathcal{C}'$ is also an optimum solution.

□

Using the properties presented above we are ready to describe the procedure that computes the value of $f(t,x_1,x_2,\ldots,x_m)$.

Case 1. If for all i it holds $x_i > 0$, then $f(t,x_1,x_2,\ldots,x_m) = f(t+1,x_1-1,x_2-1,\ldots,x_m-1)$.

Case 2. There is at least one $x_i = 0$. Let i be the smallest index for which this holds and let $v_iw_i = e_i$ be its corresponding edge. Now, to temporally cover e_i in the time window W_t we check the last appearance of all edge configurations of e_i at v_i, w_i in the time window W_t, and choose the one that minimizes the solution. Namely, if we cover e_i at time t_i using vertex v_i we add (v_i,t_i) to the solution, set the new value of x_i to $t_i - t + 1$ and also the value x_j of every edge e_j in $\gamma(e_i,t)_{v_i}$ to $t_i - t + 1$. We then recursively continue the calculation of f. At the end, we select such t_i (such edge configuration) that minimizes the value $f(t,x_1,x_2,\ldots,x_m)$.

Running Time. It is straightforward to see that the number of table entries is bounded by $T\Delta^{|E(G)|}$ which also bounds the complexity of computing them.
A second bound can be established by noticing that each entry of f, that is filled during the dynamic program, is uniquely determined by configurations of each edge. Together both bounds yield the desired complexity. □

The above algorithm solves Partial Δ-TVC in $O(Tc^{O(|E(G)|)})$, where $c = \min\{2^{d_G}, \Delta\}$. Note, that the following parameterized complexity result of Δ-TVC follows directly from our algorithm above.

Corollary 4.4.5. *For every $\Delta \geq 2$, Δ-TVC can be solved in time in $O(Tc^{O(|E(G)|)})$ where $c = \min\{2^{O(|E(G)|)}, \Delta\}$, and thus Δ-TVC is in* FPT *parameterized by $|E(G)|$.*

4.4.2 Approximation ratio better than d

In this section, we focus on approximation algorithms. In [5], the authors present the d-factor approximation algorithm (where d is the maximum vertex degree in any time-step), which uses the following idea. First optimally solve the Δ-TVC for each edge separately, which can be done in polynomial time, and then combine all the solutions. In the worst case, it may happen that the solution includes all d neighbors of a specific vertex appearance (v, t), instead of just (v, t), which is how the bound is obtained. We try to avoid this situation by iteratively covering paths with two edges (P_3-s) instead of single edges. With this approach, we force the middle vertex (called *central vertex*) to be in the temporal vertex cover, which gives us an error of at most $d - 1$. Based on this idea we can formulate our $(d - 1)$-approximation algorithm. Note in our case we require that $d \geq 3$.

Description of the algorithm

Let $\mathcal{G} = (G, \lambda)$ be an instance of Δ-TVC. We iteratively extend an initially empty set $\mathcal{X}$ to a Δ-TVC of $\mathcal{G}$ in the following way. While there is an edge $e \in E(G)$ that is not covered in some time window of $\mathcal{G}$ we have to extend $\mathcal{X}$; otherwise $\mathcal{X}$ is a Δ-TVC which we can return. We proceed in two phases:

1. While there is a 3-vertex path P that appears at some time step t and whose underlying edges are uncovered by $\mathcal{X}$ in some time window W_i with $t \in W_i$, we build an instance of the Partial Δ-TVC in the following way. Let S be the set of all time-steps t in which both edges of P appear and for which there are time windows containing that time-step in which each edge of P is uncovered.

More specifically,

$$S = \{t \in [T] \mid P \text{ appears at } t \text{ and for every } e \in E(P)$$
$$\text{there is a } W_i \text{ such that } t \in W_i \text{ and } e \text{ is uncovered in } W_i\}.$$

We subdivide S into subsets $S_1, \dots S_k$ with $k \leq T$ such that there is a gap of at least Δ between the last element of S_i and the first element of S_{i+1} (for all $i \in [k-1]$). Now we iteratively consider each S_i and construct the Partial Δ-TVC instance $\mathcal{H}$ given by $(P, {\lambda'}_i^P, \ell_i(P), h_i(P))$ with ${\lambda'}_i^P$ defined as a mapping of each edge of P to S_i, $\ell_i(P)$ is the smallest value t such that $(P, \min S_i)$ is not covered in time window W_t by $\mathcal{X}$, and $h_i(P)$ is the smallest value t such that $(P, \max S_i)$ is not covered in time window W_t by $\mathcal{X}$. We use the algorithm from Section 4.4.1 to solve such an instance, extend $\mathcal{X}$ by the union of this solution, and continue with S_{i+1}.
Note that these are the instances that bring the crucial advantage compared to the known d-approximation. To ensure our approximation guarantee, we also need to allow the inclusion of appearances of a single underlying edge in different P_3 instances for an example see Figure 4.14. This is why we restrict the temporal extent of each P_3 instance to time-steps which are impacted by the actual appearances of the respective P_3.

2. When there is no such P_3 left, we set F to be the set of edges $e \in E(G)$ that are not yet covered in some time window of $\mathcal{G}$. For $e \in F$, let S^e be the set of all time-steps in which e appears and is not covered. We subdivide S^e into subsets $S^e_1, \dots S^e_k$ with $k \leq T$ such that there is a gap of at least $2\Delta - 1$ between the last element of S^e_i and the first element of S^e_{i+1} (for all $i \in [k-1]$). Now we consider the Partial Δ-TVC instance given by $(\{e\}, {\lambda'}^e_i, \ell_i(e), h_i(e))$ with ${\lambda'}^e_i$ defined a mapping of e to S^e_i, $\ell_i(e)$ is the smallest value t such that $(e, \min S^e_i)$ is not covered in time window W_t, and $h_i(e)$ is the largest value t such that $(e, \max S^e_i)$ is not covered in time window W_t. We use the algorithm from Section 4.4.1 to solve these instances and extend $\mathcal{X}$ by the union of the solutions.

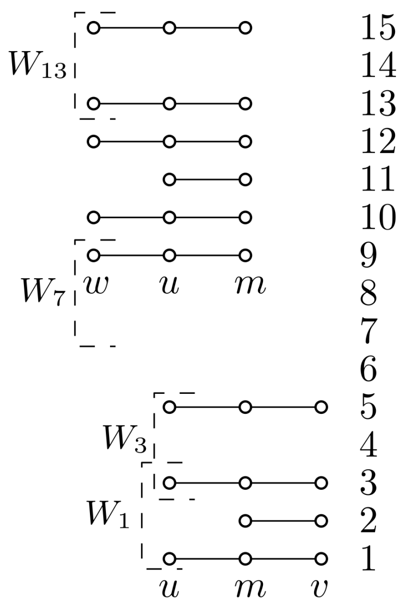


Figure 4.14: An example of a temporal graph with $\Delta = 3$ and two of the Partial Δ-TVC instances $\mathcal{H}_1 = (P_1, \lambda_1, \ell_1, h_1)$ and $\mathcal{H}_2 = (P_2, \lambda_2, \ell_2, h_2)$, generated by our algorithm. Here, $\mathcal{H}_1$ is defined as $P_1 = (u, m, v)$, $S_1 = \{1, 3, 5\}$, $\ell_1 = 1$, $h_1 = 3$, $\lambda_1(um) = \lambda_1(mv) = S_1$. Also, $\mathcal{H}_2$ is defined as $P_2 = (w, u, m)$, $S_2 = \{9, 10, 12, 13, 15\}$, $\ell_2 = 7$, $h_2 = 13$, $\lambda_2(wu) = \lambda_2(um) = S_2$. Note that $\mathcal{H}_1 \cap \mathcal{H}_2 = \emptyset$, i.e. $\mathcal{H}_1$ and $\mathcal{H}_2$ are two disjoint Partial Δ-TVC instances, while the edge um is present in both underlying paths P_1 and P_2.

Intuitively, these instances just take care of whatever remains after the first phase. We split them up temporally, just for compatibility with the input for the algorithm from Section 4.4.1.

It follows from the above construction that the produced set $\mathcal{X}$ of vertex appearances is a Δ-TVC of $\mathcal{G}$.

Running time and the approximation ratio $(d-1)$.

The number of instances considered in Phase 1 is bounded by the number of combinations of two edges in $E(G)$ multiplied by T. Similarly, the number of instances considered in Phase 2 is bounded by the number of edges in $E(G)$ multiplied by T. To find an optimal solution for each instance we use the algorithm from Theorem 4.4.2. Since each instance has at most 2 edges and the maximum degree is at most 2, the algorithm requires $O(T)$ time to solve it optimally. Therefore, the overall running time lies in $O(|E(G)|^2 T^2)$.

Now we argue the claimed approximation guarantee. For this we denote with $\mathfrak{H}$ the set of all Partial Δ-TVC instances created by our algorithm. Let $\mathcal{C}$ be a minimum Δ-TVC for $\mathcal{G}$, and $\mathcal{X}$ be the temporal vertex set computed by our algorithm. A Partial Δ-TVC instance $\mathcal{H} \in \mathfrak{H}$ is of form $\mathcal{H} = (H, \lambda'_H, \ell_H, h_H)$, where H is either a P_3 (Phase 1 instance) or an edge (Phase 2 instance).

Let us now define the set A as the set of triples $(v, t, \mathcal{H})$, where $(v, t) \in \mathcal{C}$ is a vertex appearance from the optimum solution $\mathcal{C}$, and $\mathcal{H} = (H, \lambda'_H, \ell_H, h_H) \in \mathfrak{H}$ is such Partial Δ-TVC instance, created by our algorithm, that the vertex appearance (v, t) covers one appearance of an edge of H in a time window between W_{ℓ_H} and W_{h_H}. More precisely,

$$\begin{aligned} A = \{(v,t,\mathcal{H}) \mid & (v,t) \in \mathcal{C}, \mathcal{H} \in \mathfrak{H} \wedge \\ & (v,t) \text{ covers an appearance of } e \in E(H) \\ & \text{at some time } t' \in \lambda(e) \\ & \text{in } W_i \text{ with } \ell_H(e) \leq i \leq h_H(e)\} \end{aligned}$$

From the definition of $\mathcal{X}$ and A it follows that

$$|\mathcal{X}| \leq |A|, \tag{4.3}$$

as our algorithm optimally covers every Partial Δ-TVC instance $\mathcal{H} \in \mathfrak{H}$, and, in case vertex appearances in the optimal solution $(v, t) \in \mathcal{C}$ differ from the ones chosen by our algorithm, we may need more of them to cover each $\mathcal{H}$.

To show the desired approximation guarantee we give a double-counting argument for the cardinality of A. Counting from one side we distinguish vertices in the optimal solution $\mathcal{C}$ based on what kind of Partial Δ-TVC instances $\mathcal{H} \in \mathfrak{H}$ they cover time-edges in. To count the number of elements in A from the other side we distinguish the Partial Δ-TVC instances $\mathcal{H} \in \mathfrak{H}$ by the number of vertices in a fixed optimal solution $\mathcal{C}$, that cover the edges appearing in $\mathcal{H}$. We either use the optimal amount of vertices in $\mathcal{C}$ to cover $\mathcal{H}$ or we have to use more.

More specifically, on one side we want to associate Partial Δ-TVC instances

$\mathcal{H}$ with vertex appearances $(v,t) \in \mathcal{C}$. For this, we define the following set

$$\begin{aligned}\mathfrak{H}(v,t) = \{\mathcal{H} \in \mathfrak{H} \mid (v,t) \in \mathcal{C} &\text{ covers an appearance}\\ &\text{of } e \in E(H) \text{ at time } t' \in \lambda(e)\\ &\text{in } W_i \text{ with } \ell_H(e) \le i \le h_H(e)\}.\end{aligned}$$

On the other side, we want to associate vertex appearances $(v,t) \in \mathcal{C}$ with the Partial Δ-TVC instances $\mathcal{H}$. For this, we define the following set

$$\begin{aligned}\mathcal{C}(\mathcal{H}) = \{(v,t) \in \mathcal{C} \mid (v,t) &\text{ covers an appearance}\\ &\text{of } e \in E(H) \text{ in } W_i \text{ with}\\ &\ell_H(e) \le i \le h_H(e)\}.\end{aligned}$$

Intuitively, one can think of the sets of form $\mathfrak{H}(v,t)$, for some vertex appearance (v,t), as the collection of Partial Δ-TVC instances "touching" (v,t), and $\mathcal{C}(\mathcal{H})$, for some Partial Δ-TVC instance $\mathcal{H}$, as the set of vertex appearances "touching" $\mathcal{H}$. For an example see Figure 4.15.

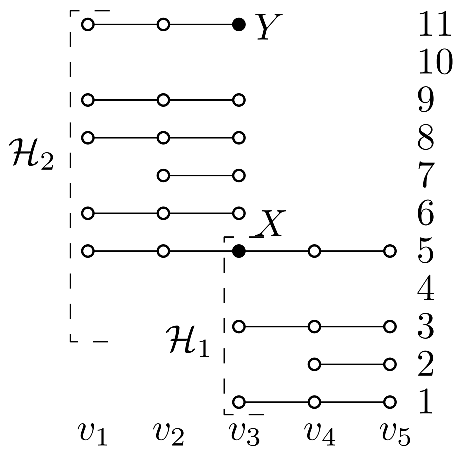


Figure 4.15: An example of a temporal graph with $\Delta = 3$ and two of the Partial Δ-TVC instances $\mathcal{H}_1 = (P_1, \lambda_1, \ell_1, h_1)$ and $\mathcal{H}_2 = (P_2, \lambda_2, \ell_2, h_2)$, generated by our algorithm. Here, $\mathcal{H}_1$ is defined as $P_1 = (v_3, v_4, v_5)$, $S_1 = \{1, 3, 5\}$, $\ell_1 = 1$, $h_1 = 3$, $\lambda_1(v_3v_4) = \lambda_1(v_4v_5) = S_1$. Also, $\mathcal{H}_2$ is defined as $P_2 = (v_1, v_2, v_3)$, $S_2 = \{5, 6, 8, 9, 11\}$, $\ell_2 = 3$, $h_2 = 9$, $\lambda_2(v_1v_2) = \lambda_2(v_2v_3) = S_2$. We have two vertex appearances $X = (v_3, 5)$ and $Y = (v_3, 11)$ from $\mathcal{C}$, and their corresponding sets $\mathfrak{H}(X) = \{\mathcal{H}_1, \mathcal{H}_2\}$, $\mathfrak{H}(Y) = \{\mathcal{H}_2\}$, $\mathcal{C}(\mathcal{H}_1) = \{X\}$, $\mathcal{C}(\mathcal{H}_2) = \{X, Y\}$.

For brevity, when speaking of a Partial Δ-TVC instance $\mathcal{H}$ constructed by our algorithm and a vertex appearance (v, t), we say that (v, t) *subcovers* an edge $e \in E(H)$ if (v, t) covers an appearance of e in W_i with $\ell_H(e) \leq i \leq h_H(e)$.

We can now begin with counting the number of elements in A. From the definition, it follows that

$$|A| = \sum_{(v,t) \in \mathcal{C}} |\mathfrak{H}(v, t)|.$$

We can split the sum into two cases, first where one vertex appearance subcovers one edge (or even a P_3) of at most $d - 1$ different Partial Δ-TVC instances, and the second when one vertex appearance subcovers one edge (or a P_3) of more than $d - 1$ different Partial Δ-TVC instances.

$$|A| = \sum_{\substack{(v,t) \in \mathcal{C} \\ |\mathfrak{H}(v,t)| \leq d-1}} |\mathfrak{H}(v, t)| + \sum_{\substack{(v,t) \in \mathcal{C} \\ |\mathfrak{H}(v,t)| > d-1}} |\mathfrak{H}(v, t)|$$

By the construction of the Partial Δ-TVC, it follows that if (v, t) has a temporal degree k, it subcovers edges in at most k different Partial Δ-TVC instances. Moreover, if a vertex appearance (v, t) subcovers two edges of a single Partial Δ-TVC instance $\mathcal{H}$ (i. e., $\mathcal{H}$ is a Phase 1 instance with a P_3 as the underlying graph and v is a central vertex of P_3), then (v, t) can subcover time-edges in one less Partial Δ-TVC instance. More specifically, if we assume that (v, t) with temporal degree k is a central vertex of at least one Partial Δ-TVC instance (i. e., subcovers both edges of at least one instance), then (v, t) can subcover edges in at most $k - 1$ different instances. Note that, since the degree of each vertex is at most d, by our assumption, this means that for all vertex appearances $(v, t) \in \mathcal{C}$ that subcover both time-edges of some Phase 1 Partial Δ-TVC instance it holds that $|\mathfrak{H}(v, t)| \leq d-1$. Therefore, for the second case to happen vertex appearance $(v, t) \in \mathcal{C}$ must subcover exactly one underlying edge of d different Partial Δ-TVC instances. For brevity we denote $D := \{(v,t) \in \mathcal{C} \mid |\mathfrak{H}(v,t)| = d\}$ Hence the following holds

$$\begin{aligned} |A| &\leq (d-1)(|\mathcal{C}| - |D|) + \sum_{(v,t)\in D} |\mathfrak{H}(v,t)| \\ &\leq (d-1)|\mathcal{C}| + \sum_{(v,t)\in D} (d-(d-1)) \\ &= (d-1)|\mathcal{C}| + |D|. \end{aligned} \tag{4.4}$$

Let us now count the number of elements in A from the other side. Again by the definition, we get that

$$|A| = \sum_{\mathcal{H}\in\mathfrak{H}} |\mathcal{C}(\mathcal{H})|.$$

Since $|A| \geq |\mathcal{X}|$ (see Equation (4.3)), we can divide A into two parts. The first one, where the solution for the partial instances computed by our algorithm coincide (in terms of cardinality) with the ones implied by $\mathcal{C}$, and the second where $\mathcal{C}$ implies suboptimal solutions. For this we define $\mathcal{X}(\mathcal{H})$ as the optimal solution for Partial Δ-TVC instance $\mathcal{H}$ calculated by our algorithm. From the construction it holds that $|\mathcal{X}| = \sum_{\mathcal{H}\in\mathfrak{H}} |\mathcal{X}(\mathcal{H})|$. We get the following

$$|A| = |\mathcal{X}| + \sum_{\mathcal{H}\in\mathfrak{H}} \left(|\mathcal{C}(\mathcal{H})| - |\mathcal{X}(\mathcal{H})|\right). \tag{4.5}$$

Combining Equations (4.4) and (4.5) we obtain

$$|\mathcal{X}| + \sum_{\mathcal{H}\in\mathfrak{H}} \left(|\mathcal{C}(\mathcal{H})| - |\mathcal{X}(\mathcal{H})|\right) \leq (d-1)|\mathcal{C}| + |D|.$$

Now, to prove our target, namely that $|\mathcal{X}| \leq (d-1)|\mathcal{C}|$, it suffices to show that

$$|D| \leq \sum_{\mathcal{H}\in\mathfrak{H}} \left(|\mathcal{C}(\mathcal{H})| - |\mathcal{X}(\mathcal{H})|\right). \tag{4.6}$$

Recall that $|\mathfrak{H}(v,t)| > (d-1)$ holds only for vertex appearances $(v,t) \in \mathcal{C}$, with temporal degree d, that cover appearances of exactly one edge from every Partial Δ-TVC $\mathcal{H} \in \mathfrak{H}(v,t)$. Furthermore, it turns out that at most one Partial Δ-TVC $\mathcal{H} \in \mathfrak{H}(v,t)$ is a Phase 2 (a single edge) instance.

Lemma 4.4.6. *Let $(v,t) \in \mathcal{C}$. There is at most one Phase 2 (single edge)* Partial *Δ-TVC instance $\mathcal{H}_1 \in \mathfrak{H}(v,t)$.*

Proof. Suppose this is not true. Then there are two different time-edges $(e,t),(f,t)$ incident to (v,t), that belong to Phase 2 Partial Δ-TVC instances $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. W.l.o.g. suppose that our algorithm created instance $\mathcal{H}_1$ before $\mathcal{H}_2$. By construction the algorithm creates a Phase 2 Partial Δ-TVC instance whenever there is no uncovered P_3 left in the graph. But, when creating $\mathcal{H}_1$ both time-edges $(e,t),(f,t)$ are uncovered in some time windows W_e and W_f, respectively, where $t \in W_e \cap W_f$. Therefore $(e,t),(f,t)$ form an uncovered P_3 path at time-step t, which would result in the algorithm combining them into one single Phase 1 Partial Δ-TVC instance. □

We now proceed with the proof of Equation (4.6).

Lemma 4.4.7. *For D and $\sum_{\mathcal{H}\in\mathfrak{H}} (|\mathcal{C}(\mathcal{H})| - |\mathcal{X}(\mathcal{H})|)$ defined as above the following holds*

$$|D| \leq \sum_{\mathcal{H}\in\mathfrak{H}} (|\mathcal{C}(\mathcal{H})| - |\mathcal{X}(\mathcal{H})|) . \tag{4.7}$$

Proof. We show the statement by modifying the solutions implied by $\mathcal{C}$ on Phase 1 partial Δ-TVC instances (paths of length 3). More specifically, for each $(v,t) \in D$ and every Phase 1 partial Δ-TVC instance $\mathcal{H} \in \mathfrak{H}(v,t)$, we find a vertex appearance in $\mathcal{C}(\mathcal{H})$ which is "disposable in the solution". We refer to a set of vertex appearances including (v,t) as disposable in the solution of $\mathcal{H} \in \mathfrak{H}(v,t)$ if we can modify the solution $\mathcal{C}(\mathcal{H})$ by decreasing its size by the size of that set. In our process we not only show that (v,t) is disposable in the solution of $\mathcal{H}$, but that the set of all vertex appearances of v in $\mathcal{C}(\mathcal{H})$ (meaning all vertex appearances of form (v,t_i)) are disposable in the solution of $\mathcal{H}$. To produce a solution $\widetilde{\mathcal{C}(\mathcal{H})}$ of $\mathcal{H}$ which proves that certain vertex appearances are disposable in the solution $\mathcal{C}(\mathcal{H})$ of $\mathcal{H}$ we perform the following three steps. Firstly, we modify the solution $\mathcal{C}(\mathcal{H})$ to only include vertex appearances at time-steps when the entire P_3 underlying $\mathcal{H}$ appears. Secondly, we ensure that the included vertex appearance corresponds to the central vertices of the P_3 underlying $\mathcal{H}$. Thirdly, we remove unnecessary vertex appearances, producing the desired solution. The process is depicted in Figure 4.16. While determining

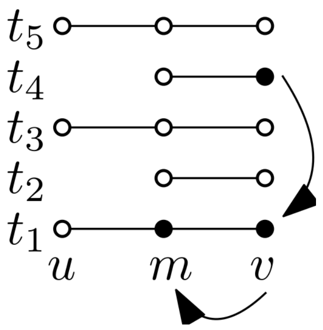


Figure 4.16: An illustration of a modification of a solution $\mathcal{C}(\mathcal{H})$ presented in Lemma 4.4.7.
The figure represents a PARTIAL Δ-TVC instance $\mathcal{H}$, where the vertex appearance $(v, t_4) \in \mathcal{C}$. We first modify the solution $\mathcal{C}(\mathcal{H})$ by replacing (v, t_4) with (v, t_1), and then further replace it with the central vertex (m, t_1).

disposable vertex appearances does not correspond to an injection on D we ensure that no single disposable vertex appearance is used for more than two elements of D, which proves to be sufficient in demonstrating the desired bound.

We start by inspecting vertex appearances $(v, t) \in D$ in an order increasing with t. Recall that $(v, t) \in D$ implies that there is no Phase 1 PARTIAL Δ-TVC instance in $\mathfrak{H}(v, t)$ in which the associated temporal graph contains two edge appearances incident to v at t, because otherwise $|\mathfrak{H}(v, t)| \leq d - 1$. Since $d - 1 \geq 2$, together with Lemma 4.4.6 this implies that there are at least two Phase 1 PARTIAL Δ-TVC instances in $\mathfrak{H}(v, t)$ for each of which v is incident to only one of the underlying edges. Notice that here it is important that $d \geq 3$.

Let $\mathcal{H}$ be a Phase 1 subinstance from $\mathfrak{H}(v, t)$. W.l.o.g. we denote with $P = (u, m, v)$ the underlying path of $\mathcal{H}$, and let ℓ_P and h_P be the lowest and the highest time-steps in the definition of $\mathcal{H}$, respectively. Note that vertex appearances in the intersection $\mathcal{C}(\mathcal{H}) \cap D$ are of the form $(v, t_i), (u, t_j)$ for some t_i, t_j (if they exist). Suppose that there are $k_v \geq 1$ vertices of form (v, t_i) in $\mathcal{C}(\mathcal{H})$. Our goal is to create the modified solution $\widetilde{\mathcal{C}(\mathcal{H})}$ of $\mathcal{H}$ where $|\widetilde{\mathcal{C}(\mathcal{H})}| \leq |\mathcal{C}(\mathcal{H})| - k_v$. We start by setting $\widetilde{\mathcal{C}(\mathcal{H})} = \mathcal{C}(\mathcal{H})$. We first iterate through every vertex appearance $(u, t') \in \mathcal{C}(\mathcal{H})$ (increasingly in t'), and do the following. We set $T_{\mathcal{H}} = [\ell_P, h_P + \Delta - 1] \cap [t' - \Delta + 1, t' + \Delta - 1]$ to be the interval of time-steps t'' for which $\mathcal{H}$ is defined, such that $|t'' - t'| \leq \Delta$. By the construction of Phase 1 subinstances, both edges um and mv appear at some time step t^* in $T_{\mathcal{H}}$. We chose $t^* \in T_{\mathcal{H}}$ to be maximal such that (x, t^*)

covers an appearance of the edge um in the earliest time window $t^* \in W_i$ of $\mathcal{H}$, where the edge um is not yet covered by vertex appearances in $\widetilde{\mathcal{C}(\mathcal{H})} \setminus (u, t')$. Intuitively, when removing (u, t') from $\widetilde{\mathcal{C}(\mathcal{H})}$ we may get some time-windows $W_i, W_{i+1}, W_{i+2} \ldots$, where the edge um is not yet covered. We now set t^* to be the maximum value in W_i when both um and mv appear. If such a (u, t^*) exists we replace (u, t') by (u, t^*), otherwise we identify (u, t') as disposable in the solution for $\mathcal{H}$. We then further modify $\widetilde{\mathcal{C}(\mathcal{H})}$ by replacing each (u, t^*) from the previous step with (m, t^*). We now repeat the same procedure for all $(v, t') \in \widetilde{\mathcal{C}(\mathcal{H})}$. After the first two steps are finished $\widetilde{\mathcal{C}(\mathcal{H})}$ consists only of the vertex appearances of the central vertex m of P. We now continue with the final step and iterate through the vertex appearances (m, t) in the order of increasing t, delete unnecessary vertices from the solution $\widetilde{\mathcal{C}(\mathcal{H})}$ for $\mathcal{H}$ and mark them as disposable. This happens if (m, t) has been added to $\widetilde{\mathcal{C}(\mathcal{H})}$ multiple times, when considering different vertex appearances, or it can be removed from $\widetilde{\mathcal{C}(\mathcal{H})}$ without changing the fact that it is a solution. In both cases, any appearance of the vertex v from the set D which is associated with the unnecessary vertex appearance (m, t^*) is disposable in the solution for $\mathcal{H}$.

We now have to show that $\widetilde{\mathcal{C}(\mathcal{H})}$ is indeed a solution for $\mathcal{H}$. To prove this we have to make sure that exchanging each vertex appearance $(u, t') \in \mathcal{C}(\mathcal{H})$ (resp. $(v, t') \in \mathcal{C}(\mathcal{H})$) with (u, t^*) (resp. (v, t^*)) and then finally with (m, t^*) in the first two steps of the modification returns a valid solution. Suppose for the contradiction that the edge mv appears at time t and that there is a time window with $t \in W_i$ where mv is uncovered in W_i by our final solution $\widetilde{\mathcal{C}(\mathcal{H})}$. Using the definition of $\mathcal{H}$ and the fact that $\widetilde{\mathcal{C}(\mathcal{H})}$ contains only the appearances of the central vertex of P it follows that also um appears at time t and is uncovered in W_i. In the initial solution, um and mv were both covered in W_i. If they were both covered by the same vertex appearance, it must have been the appearance of the central vertex which would remain in our final solution. Therefore, the edges are covered by two different vertex appearances. Denote them with (u, t_u) and (v, t_v), respectively. Note that $t_u, t_v \in W_i$. Therefore, $|t_v - t_u| \leq \Delta$. In our procedure we first considered (u, t_u) that was replaced by (u, t_u^*) and then by (m, t_u^*), where the edge um was uncovered by $\widetilde{\mathcal{C}(\mathcal{H})} \setminus (u, t_u)$ in a time window W_j^u. It also holds that t_u^* is the maximum value of W_j^u when both

um and mv occur. Since um and mv are uncovered in W_i it must be true also that $t_u^* \notin W_i$, even more $t_u^* < i$. Next, the procedure considered (v, t_v), and replaced it by (v, t_v^*) and then further by (m, t_v^*). Note here that t_v^* is the maximum value in some time window W_j^v where both um and mv appear and are uncovered by the modified solution $\widetilde{\mathcal{C}(\mathcal{H})} \setminus (v, t_v)$, where $\widetilde{\mathcal{C}(\mathcal{H})}$ already includes the vertex appearance (m, t_u^*). This implies that $t_v^* \notin W_j^u$ and furthermore, since $t_u, t_v \in W_i$ and $t_u - t_u^* \leq \Delta$ it follows that $t_v^* \in W_i$. Therefore, um and mv cannot be uncovered in W_i.

What remains to show is that our modified solution is of the correct size. Namely $|\widetilde{\mathcal{C}(\mathcal{H})}| \leq |\mathcal{C}(\mathcal{H})| - k_v$, where k_v is the number of vertex appearances of v in $\mathcal{C}(\mathcal{H})$. We show this by proving that each vertex $(v, t_i) \in \mathcal{C}(\mathcal{H}) \cap D$ is disposable in the solution of $\mathcal{H}$. Remember, our process first inspects all the vertex appearances of form $(u, t_u) \in \mathcal{C}(\mathcal{H})$ and replaces them with the corresponding (m, t_u^*). Since in the original solution $\mathcal{C}(\mathcal{H})$ the whole underlying path P of $\mathcal{H}$ was covered, it follows that after performing the first two steps for the vertices (u, t_u) the edge um remains covered in $\mathcal{H}$. Even more, because each (u, t_u) was exchanged by a corresponding appearance of the central vertex m at time t_u^* when both um and mv appear, it follows that the edge mv is covered in $\mathcal{H}$. Therefore, we can denote all $(v, t_i) \in \mathcal{C}(\mathcal{H})$ as disposable in the solution of $\mathcal{H}$.

Overall, this means that for every vertex appearance $(v, t) \in D$ and each of at least two Phase 1 Partial Δ-TVC instances in $\mathcal{H} \in \mathfrak{H}(v, t)$, we can find one disposable vertex appearance in $\mathcal{C}(\mathcal{H})$. On the other hand, these found disposable vertex appearances coincide for at most two vertex appearances in D (this is true as D can admit vertex appearances of both endpoints of the underlying path of a Partial Δ-TVC instance $\mathcal{H}$). By Hall's theorem [72] this means we can match each vertex appearance in D to a disposable vertex appearance in $\mathcal{C}$, yielding that $|D|$ is upper-bounded by $\sum_{\mathcal{H} \in \mathfrak{H}} (|\mathcal{C}(\mathcal{H})| - |\mathcal{X}(\mathcal{H})|)$ as desired. □

All of the above shows the desired approximation result.

Theorem 4.4.8. *For every $\Delta \geq 2$ and $d \geq 3$, Δ-TVC can be $(d-1)$-approximated in time $O(|E(G)|^2 T^2)$.*

4.4.3 An FPT algorithm with respect to the solution size

Our final result settles the complexity of Δ-TVC from the viewpoint of parameterized complexity theory [34, 39, 108] with respect to the standard parameterization of the size of an optimum solution.

Theorem 4.4.9. *For every $\Delta \geq 2$, Δ-TVC can be solved in $O((2\Delta)^k T n^2)$ time, where k is the size of an optimum solution. In particular Δ-TVC is in* FPT *parameterized by k.*

Proof. The algorithm starts by selecting an edge $e \in E(G)$ with an uncovered appearance in an arbitrary Δ-time window W_t and selects one of (at most) Δ vertex appearances of either endpoint to cover it. By covering one edge at some time point, we also cover some other edges in some other time windows, these are also excluded from the subsequent search. This step is repeated until after k-steps the algorithm stops and checks if selected vertices form a sliding Δ-time window temporal vertex cover of $\mathcal{G}$.

The algorithm builds at most $(2\Delta)^k$ sets, for each of which checking if it is a temporal vertex cover takes $O(Tn^2)$ time. Therefore the running time of the algorithm is $O((2\Delta)^k T n^2)$. □

4.5 Concluding remarks

In this chapter, we presented a comprehensive analysis of the complexity of temporal vertex cover in small-degree graphs. We showed that Δ-TVC is NP-hard on the underlying paths (and cycles) already for $\Delta = 2$, while TVC on these underlying graphs can be solved in polynomial time. An open question that arises is for which values of k we have that $(T - k)$-TVC is polynomial-time solvable, where T is the lifetime of the given temporal graph.

Additionally, we provided a Polynomial-Time Approximation Scheme (PTAS) for Δ-TVC on temporal lines and cycles, complementing the hardness result. We present also an exact dynamic programming algorithm, that we then use in the $d - 1$ approximation algorithm (where d is the maximum vertex degree in any time step), which improves the d-approximation algorithm presented by Akrida et al. [5]. Furthermore, we settled the complexity of Δ-TVC from the viewpoint of parameterized complexity theory, showing that Δ-TVC is fixed-parameter tractable when parameterized by the size of an optimum solution.

4.5.1 Experimental results

In the academic year 2022/2023, Sophia Heck, then a master's student at Heidelberg University and currently a PhD student at the University of Vienna, worked on her master's thesis titled *Approximation Algorithms for the Minimum (Sliding Window) Temporal Vertex Cover Problem*. She successfully implemented our $d-1$ approximation algorithm and compared it to the d-approximation algorithm from Akrida et al. As expected, the $d-1$ approximation algorithm outperforms the d-approximaiton one. Her thesis is accessible on the following website: schulzchristian.github.io.

CHAPTER 5

The Complexity of Computing Optimum Labelings for Temporal Connectivity

This chapter is based on a joint work with George B. Mertzios, Hendrik Molter and Paul G. Spirakis. The preliminary results were presented in the Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS) 2022 [87]. The full paper, containing our detailed results, is accessible as a preprint on ArXiv [88]. As of Autumn 2023, it is also under the submission process in the Journal of Computer and System Sciences (JCSS).

The major part of the proof for Theorem 5.4.2 was contributed by Hendrik Molter; therefore, I have chosen not to present it in detail in this chapter. The complete proof, including all the details, is available in our full paper, while I provide the main ideas also here.

5.1 Introduction

Motivated by the need to restrict the spread of the epidemic, Enright et al. [42] studied the problem of removing the smallest number of time-labels from a given temporal graph such that every vertex can only temporally reach a limited number of other vertices. Deligkas et al. [36] studied the problem of accelerating the spread of information for a set of sources to all vertices in a temporal graph, by only using delaying operations, i. e., by shifting specific time-labels to a later time slot. The problems studied by Deligkas et al. [36] are related but orthogonal to our temporal connectivity problems.

The time-labels of an edge e in a temporal graph indicate the discrete units of time (e.g., days, hours, or even seconds) in which e is active. However, in many real dynamic systems, e.g., in synchronous mobile distributed systems that operate in discrete rounds, or in unstable chemical or physical structures, maintaining an edge over time requires energy and thus comes at a cost. One natural way to define the *cost* of the whole temporal graph (G, λ) is the *total number* of time-labels used in it, i. e., the total cost of (G, λ) is $|\lambda| = \sum_{e \in E} |\lambda_e|$. In this chapter, we study *temporal design* problems of undirected temporally connected graphs, where a temporal graph is temporally connected if there exists a temporal path between each pair of vertices. The basic setting of these optimization problems was introduced by Mertzios et al. [98] and is as follows: given an undirected graph G, what is the smallest number $|\lambda|$ of time-labels that we need to assign to the edges of G such that (G, λ) is temporally connected? As it turns out, this basic problem can be optimally solved in polynomial time, thus answering a conjecture made by Akrida et al. [4]. However, by exploiting the temporal dimension, the problem becomes more interesting and meaningful in its following variations, which we investigate in this chapter. First, we consider the problem variation where we are given along with the input also an upper bound of the allowed *age* (i. e., maximum label) of the obtained temporal graph (G, λ). This age restriction is sensible in more pragmatic cases, where delaying the latest arrival time of any temporal path incurs further costs, e.g., when we demand that all agents in a safety-critical distributed network are synchronized as quickly as possible, and with the smallest possible number of communications among them. Second, we consider problem variations where the aim is to have a temporal path between any pair of "important" vertices which lie in a subset $R \subseteq V$, which we call the *terminals*. For a detailed definition of the studied problems, we refer to Section 5.2.

Here it is worth noting that the latter relaxation of temporal connectivity resembles the problem Steiner Tree in static (i. e., non-temporal) graphs. Given a connected graph $G = (V, E)$ and a set $R \subseteq V$ of terminals, Steiner Tree asks for a smallest-sized subgraph of G which connects all terminals in R. Clearly, the smallest subgraph sought by Steiner Tree is a tree. As it turns out, this property does not carry over to the temporal case. Consider for example an arbitrary graph G and a terminal set $R = \{a, b, c, d\}$ such that G contains an induced cycle on four vertices a, b, c, d; that is, G contains the edges ab, bc, cd, da but not the edges ac or bd. Then, it is not hard to check that the only way to add the smallest number of time-labels such that all vertices of R are temporally connected is to assign one label to each edge of the cycle on a, b, c, d, e.g., $\lambda(ab) = \lambda(cd) = 1$ and $\lambda(bc) = \lambda(cd) = 2$. The main underlying reason for this difference with the static problem Steiner Tree is that temporal connectivity is *not transitive* and *not symmetric:* if there exist temporal paths from u to v, and from v to w, it is not a priori guaranteed that a temporal path from v to u, or from u to w exists.

Temporal network design problems have already been considered in previous works. Mertzios et al. [98] proved that it is APX-hard to compute a minimum-cost labeling for temporally connecting an input *directed* graph G, where the age of the graph is upper-bounded by the diameter of G. This hardness reduction was strongly facilitated by the careful placement of the edge directions in the constructed instance, in which every vertex was reachable in the static graph by only constantly many vertices. Unfortunately, this cannot happen in an undirected connected graph, where every vertex is reachable by all other vertices. Later, Akrida et al. [4] proved that it is also APX-hard to *remove* the largest number of time-labels from a given temporally connected (undirected) graph (G, λ), while still maintaining temporal connectivity. In this case, although there are no edge directions, the hardness reduction was strongly facilitated by the careful placement of the initial time-labels of λ in the input temporal graph, in which every pair of vertices could be connected by only a few different temporal paths, among which the solution had to choose. Unfortunately, this cannot happen when the goal is to add time-labels to an undirected connected graph, where there are potentially multiple ways to temporally connect a pair of vertices (even if we upper-bound the largest time-label by the diameter).

Summarizing, the above technical difficulties seem to be the reason why the problem of *adding* the minimum number of time-labels with an age-restriction to an *undirected* graph to achieve temporal connectivity remained open until now. In this work, we overcome these difficulties by developing a hardness reduction from a variation of the problem MAX XOR SAT (see Theorem 5.3.6 in Section 5.3) where we manage to add the appropriate (undirected) edges among the variable-gadgets such that simultaneously (i) the distance between any two vertices from different variable gadgets remains small (constant) and (ii) there is no shortest path between two vertices of the *same* variable gadget that leaves this gadget.

Our contribution and road-map. In the first part of this chapter, in Section 5.3, we present our results on MIN. AGED LABELING (MAL). This problem is the same as ML, with the additional restriction that we are given along with the input an upper bound on the allowed *age* of the resulting temporal graph (G, λ). Using a technically involved reduction from a variation of MAX XOR SAT, we prove that MAL is NP-complete on undirected graphs, even when the required maximum age is equal to the diameter d_G of the input static graph G.

In the second part of this chapter, in Section 5.4, we present our results on the Steiner tree versions of the problem, namely on MIN. STEINER LABELING (MSL) and MIN. AGED STEINER LABELING (MASL). The difference of MSL from ML is that, here, the goal is to have a temporal path between any pair of "important" vertices which lie in a given subset $R \subseteq V$ (the *terminals*). In Section 5.4.1 we prove that MSL is NP-complete by a reduction from VERTEX COVER, the correctness of which requires showing structural properties of MSL. Here it is worth recalling that, as explained above, the classical problem STEINER TREE on static graphs is *not* a special case of MSL, due to the requirement of strictly increasing labels in a temporal path. Furthermore, we would like to emphasize here that, as temporal connectivity is neither transitive nor symmetric, a straightforward NP-hardness reduction from STEINER TREE to MSL does not seem to exist. For example, as explained above, in a graph that contains a C_4 with its four vertices as terminals, labeling a Steiner tree is sub-optimal for MSL.

In Section 5.4.2 we prove that MASL is W[1]-hard even with respect to the number of time-labels of the solution. This also implies that MASL is W[1]-hard with respect to the number $|R|$ of terminals, since the number of time-labels in the solution is a larger parameter than the number $|R|$ of terminals.

Finally, we complete the picture by providing some auxiliary results in our preliminary Section 5.2. More specifically, in Section 5.2.1 we prove that ML and the analogue minimization versions of ML and MAL on directed acyclic graphs are solvable in polynomial time.

For an easier overview of the area, we also outline all of the known and new results in Table 5.1.

Graph restrictions	Age non-restricted	Age restricted
Directed graphs	open	APX-hard [98]
Directed acyclic graphs	poly. time solvable (see Theorem 5.2.6)	poly. time solvable (see Theorem 5.2.6)
Undirected cycles	poly. time solvable (see Theorem 5.2.5)	poly. time solvable (see Lemma 5.2.2)
Undirected graphs	poly. time solvable (see Theorem 5.2.5)	NP-hard (see Theorem 5.3.6)
Steiner labeling, undirected graphs	NP-complete, FPT w.r.t. $|R|$ (see Theorems 5.4.1 and 5.4.2)	W[1]-hard w.r.t. $|\lambda|, |R|$ (see Theorem 5.4.3)

Table 5.1: An overview of previously known results, our results, and open problems.

5.2 Preliminaries and notation

As we mentioned in the introduction, a temporal graph (G, λ) is *temporally connected* if, for every pair of vertices $u, v \in V(G)$, there exists a temporal path P_1 from u to v and a temporal path P_2 from v to u. Furthermore, given a set of terminals $R \subseteq V(G)$, the temporal graph (G, λ) is *R-temporally connected* if, for every pair of vertices $u, v \in R$, there exists a temporal path from u to v and a temporal path from v to u; note that P_1 and P_2 can also contain vertices from $V \setminus R$. Now we provide the formal definitions of the four studied decision problems.

MIN. LABELING (ML) [98] **Input:** A static graph $G = (V, E)$ and $k \in \mathbb{N}$. **Question:** Does there exist a temporally connected temporal graph (G, λ), where $\|\lambda\| \leq k$?	MIN. AGED LABELING (MAL) [98] **Input:** A static graph $G = (V, E)$ and two integers $a, k \in \mathbb{N}$. **Question:** Does there exist a temporally connected temporal graph (G, λ), where $\|\lambda\| \leq k$ and $\alpha(\lambda) \leq a$?
MIN. STEINER LABELING (MSL) **Input:** A static graph $G = (V, E)$, a subset $R \subseteq V$ and $k \in \mathbb{N}$. **Question:** Does there exist a temporally R-connected temporal graph (G, λ), where $\|\lambda\| \leq k$?	MIN. AGED STEINER LABELING (MASL) **Input:** A static graph $G = (V, E)$, a subset $R \subseteq V$, and integers $a, k \in \mathbb{N}$. **Question:** Does there exist a temporally R-connected temporal graph (G, λ), where $\|\lambda\| \leq k$ and $\alpha(\lambda) \leq a$?

Note that, for both problems MAL and MASL, whenever G is not connected or the input age bound a is strictly smaller than the diameter d of G, the answer is NO. Thus, we always assume in the remainder of the chapter that G is a connected graph and $a \geq d$, where d is the diameter of the input graph G. For simplicity of the presentation, we denote by $\kappa(G, d)$ the smallest number k for which (G, k, d) is a YES instance for MAL.

Observation 5.2.1. *For every graph G with n vertices and diameter d, we have that $\kappa(G, d) \leq n(n-1)$.*

Proof. For every vertex v of $G = (V, E)$, consider a BFS tree T_v rooted at v, while every edge from a vertex $u \neq v$ to its parent in T_v is assigned the time-label $dist(v, u)$, i. e., the length of the shortest path from v to u in G. Note that each of these time-labels is smaller than or equal to the diameter d of G. Clearly, each BFS tree T_v assigns in total $n - 1$ time-labels to the edges of G, and thus the union of all BFS trees T_v, where $v \in V$, assign in total at most $n(n-1)$ labels to the edges of G. □

The next lemma shows that the upper bound of Observation 5.2.1 is asymptotically tight as, for cycle graphs C_n with diameter d, we have that $\kappa(C_n, d) = \Theta(n^2)$.

Similar results are already known for the directed cycles [98].

Lemma 5.2.2. *Let C_n be a cycle on n vertices, where $n \neq 4$, and let d be its diameter. Then*

$$\kappa(C_n, d) = \begin{cases} d^2, & \text{when } n = 2d \\ 2d^2 + d, & \text{when } n = 2d + 1. \end{cases}$$

Proof. Let $V(C_n) = \{v_1, v_2, \ldots, v_n\}$ be the vertices of C_n. In the following, if not specified otherwise, all subscripts are considered modulo n. We distinguish two cases, depending on the parity of n.

Case 1: n is odd. Let $n = 2d + 1$. Then, for each vertex $v_i \in V(C_n)$, there are exactly two distinct vertices v_{i+d} and v_{i-d} at distance d from v_i. In particular, there exists a unique path of length d from v_i to v_{i+d}, and thus the only way that a temporal path (with labels at most d) can exist from v_i to v_{i+d} is that the jth edge (for every $j = 1, \ldots, d$) of the unique path of length d from v_i to v_{i+d} contains the label j. Due to symmetry, by just considering every vertex v_i of C_n, it follows that every edge of C_n must contain each of the labels $1, 2, \ldots, d$. Therefore $\kappa(C_n, d) \geq nd = 2d^2 + d$.

Conversely, in the labeling of C_n, where every edge contains every label in $\{1, 2, \ldots, d\}$, clearly the age of the temporal graph is d and there exists a temporal path from every vertex to every other vertex. Therefore $\kappa(C_n, d) \leq nd = 2d^2 + d$, and thus $\kappa(C_n, d) = 2d^2 + d$ when $n = 2d + 1$.

Case 2: n is even. Let $n = 2d$. Then, for each vertex $v_i \in V(C_n)$, there is exactly one vertex v_{i+d} at distance d from v_i, and exactly two distinct vertices v_{i+d-1} and v_{i-d+1} on distance $d - 1$ from v_i. In particular, there exists a unique path of length $d - 1$ from v_i to v_{i+d-1}, and thus the only way that a temporal path (with labels at most d) can exist from v_i to v_{i+d-1} is that the jth edge (for every $j = 1, \ldots, d - 1$) of the unique path of length $d - 1$ from v_i to v_{i+d-1} contains the label j or the label $j + 1$.

We will now prove that, without loss of generality, for every two consecutive edges $v_{i-1}v_i$ and v_iv_{i+1}, the total number of labels of these two edges is at least d, i. e., $|\lambda(v_{i-1}v_i)| + |\lambda(v_iv_{i+1})| \geq d$. Suppose otherwise that $|\lambda(v_{i-1}v_i)| + |\lambda(v_iv_{i+1})| \leq d - 1$. Then there exists some $a \in \{1, 2, \ldots, d\}$ such that neither of the edges $v_{i-1}v_i$ and

v_iv_{i+1} contains label a. First, let $a = 1$. Then any temporal path from v_i to v_{i+d} will have to start with the label at least 2, and thus cannot arrive at v_{i+d} by time d, a contradiction. Second, let $a = d$. Similarly, any temporal path from v_{i+d} to v_i will have to arrive at v_i by time $d-1$ at the latest. However, this is not possible, as the distance between v_{i+d} and v_i in C_n is d, a contradiction.

Now let $2 \leq a \leq d-1$. Then, the only way that a temporal path (with labels at most d) can exist from vertex v_{i-a} to vertex $v_{i-a+d-1}$ is that the edges $v_{i-1}v_i$ and v_iv_{i+1} contain the label $a+1$ and the label $a+2$, respectively, as both these edges cannot contain label a by assumption. Similarly, the only way that a temporal path (with labels at most d) can exist from vertex v_{i-a+1} to vertex v_{i-a+d} is that the edges $v_{i-1}v_i$ and v_iv_{i+1} contain the label $a-1$ and the label $a+1$, respectively. By symmetry it follows that edge v_iv_{i+1} also contains label $a-1$ (by just considering vertices v_{i+a} to vertex $v_{i+a-d+1}$). That is, for the two consecutive edges $v_{i-1}v_i$ and v_iv_{i+1} we have that

$$a-1, a+1 \in \lambda(v_{i-1}v_i) \cap \lambda(v_iv_{i+1}) \tag{5.1}$$

Summarizing, consider two consecutive edges $v_{i-1}v_i$ and v_iv_{i+1}. The union $\lambda(v_{i-1}v_i) \cup \lambda(v_iv_{i+1})$ of their labels always contains labels 1 and d. The only possibility that this union is missing some label a is that both $\lambda(v_{i-1}v_i)$ and $\lambda(v_iv_{i+1})$ contain both labels $a-1$ and $a+1$. Furthermore, it follows that it is not possible that two consecutive labels $a, a+1$ miss from the union $\lambda(v_{i-1}v_i) \cup \lambda(v_iv_{i+1})$. Therefore $|\lambda(v_{i-1}v_i)| + |\lambda(v_iv_{i+1})| \geq d$. Thus, as there are $\frac{n}{2}$ disjoint pairs of consecutive edges, it follows that $\kappa(C_n, d) \geq \frac{n}{2}d = d^2$.

Conversely, consider the labeling where, for every $i = 1, \ldots, d$, the edge $v_{2i-1}v_{2i}$ (resp. the edge $v_{2i}v_{2i+1}$) contains all the odd (resp. all the even) labels within the set $\{1, 2, \ldots, d\}$. It is straightforward to check that, with this labeling, there exists a temporal path (with the maximum label at most d) from every vertex to every other vertex. Therefore $\kappa(C_n, d) \leq \frac{n}{2}d = d^2$, and thus $\kappa(C_n, d) = d^2$. $\square$

5.2.1 Polynomial-time algorithms for ML

As a first warm-up, we study the problem ML, where no restriction is imposed on the maximum allowed age of the output temporal graph. It is already known by Akrida et al. [4] that any undirected graph can be made temporally connected by adding at most $2n-3$ time-labels, while for trees $2n-3$ labels are also necessary. Moreover, it was conjectured that every graph needs at least $2n-4$ time-labels [4]. Here we prove their conjecture true by proving that, if G contains (resp. does not contain) the cycle C_4 on four vertices as a subgraph, then (G, k) is a YES instance of ML if and only if $k \geq 2n-4$ (resp. $k \geq 2n-3$). The proof is done via a reduction to the gossip problem [23] (for a survey on gossiping see also [75]).

The related problem of achieving temporal connectivity by assigning to every edge of the graph at most one time-label, has been studied by Göbel et al. [69], where the relationship with the gossip problem has also been drawn. Contrary to ML, this problem is NP-hard [69]. That is, the possibility of assigning two or more labels to an edge makes the problem computationally much easier.

In the gossip problem, we have n agents from a set A. In the beginning, every agent $x \in A$ holds its own secret. The goal is that each agent eventually learns the secret of every other agent. This is done by producing a sequence of unordered pairs (x, y), where $x, y \in A$ and each such pair represents one phone call between the agents involved, during which the two agents exchange all the secrets they currently know. There are different variations of the gossiping problem. We are interested in the following one: an agent can place a call only to a specific subset of agents from A. We can represent this problem using a graph $G = (V, E)$, where for each agent $x \in A$ we introduce a vertex $v_x \in V$ and for every allowed phone call between agents x and y we add an edge $v_x v_y$ to the set of edges E of G. We then want to find a sequence of edges of minimum length, where the ℓ-th element $e = v_x v_y$ in the sequence represents the ℓ-th phone call in our gossiping problem, during which agents x and y exchange all the information they posses. The end goal is that after all the calls occur all agents know all the secrets.

The above gossip problem is naturally connected to ML. The only difference between the two problems is that, in gossip protocols, all calls are non-concurrent, while in ML we allow concurrent temporal edges, i. e., two or more edges can appear at the same time slot t. Therefore, in order to transfer the known results from gossip to ML, it suffices to prove that in ML we can equivalently consider solutions with non-concurrent edges (see Lemma 5.2.4).

From the graph G corresponding to a gossip problem, and a sequence of calls $\mathcal{C} = c(1), c(2), \ldots, c(m)$ we build a temporal graph $\mathcal{G}_{\mathcal{C}} = (G, \lambda)$ using the following procedure. For every phone call $c(i) \in \mathcal{C}$ between its two corresponding agents x_i, y_i we add the label i to the edge $(v_{x_i} v_{y_i})$ of $\mathcal{G}_{\mathcal{C}}$. In the end, the labeling λ is completely determined by the sequence of phone calls.

Observation 5.2.3. *If the sequence $c(1), c(2), \ldots, c(m)$ of m phone calls results in all agents knowing all secrets, then the above construction produces a temporally connected temporal graph $\mathcal{G}_{\mathcal{C}} = (G, \lambda)$ with $|\lambda| = m$.*

Now note that the temporal graph $\mathcal{G}_{\mathcal{C}}$ produced by the above procedure has the special property that, for every time-label $t = 1, 2, \ldots, m$, there exists exactly one edge labeled with t. In the next lemma, we prove the reverse statement of Observation 5.2.3.

Lemma 5.2.4. *Let (G, λ) be an arbitrary temporally connected temporal graph with $|\lambda| = m$ time-labels in total. Then there exists a sequence $c(1), c(2), \ldots, c(m)$ of m phone calls that results in all agents knowing all secrets.*

Proof. For each time step $t \in \{1, 2, \ldots, \alpha(G, \lambda)\}$ we order edges from E_t in an arbitrary way, so we get $e_1^t, e_2^t, \ldots, e_{k_t}^t$, where $e_i^t \in E_t$. We then combine all the orders into an ordering $O_E = E_1, E_2, \ldots, E_{\alpha(G,\lambda)} = e_1^1, e_2^1, \ldots, e_{k_1}^1, e_1^2, \ldots, e_{k_{\alpha(G,\lambda)}}^{\alpha(G,\lambda)}$ of all temporal edges. We now create a new labeling λ' of G, where the i-th edge in the ordering O_E receives the label i. This results in a temporal graph (G, λ'), where each label occurs in exactly one edge. Note that every temporal path in (G, λ) corresponds to a temporal path in (G, λ') with the same sequence of edges, and vice versa.

Finally, we create the required sequence of phone calls as follows: for every $i = 1, 2, \ldots, m$, if (G, λ') contains the edge e with time-label i, we add a phone call $c(i)$ between the two endpoints of the edge e. Since both (G, λ) and (G, λ')

are temporally connected, it follows that the sequence $c(1), c(2), \ldots, c(m)$ of calls results in every agent knowing every secret. $\square$

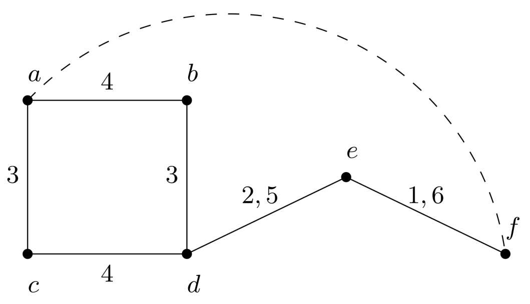
Now denote with $f(n)$ the minimum number of calls needed to complete gossiping among a set A of n agents, where only a specific set of pairs of agents $B \subseteq \binom{A}{2}$ are allowed to make a direct call between each other. Let $G_0 = (A, B)$ be the (static) graph having the agents in A as vertices and the pairs of B as edges. Then it is known by Bumby [23] that, if G_0 contains a C_4 as a subgraph then $f(n) = 2n - 4$, while otherwise $f(n) = 2n - 3$. Therefore the next theorem follows by Observation 5.2.3 and Lemma 5.2.4 and by the results of Bumby [23].

Theorem 5.2.5. *Let $G = (V, E)$ be a connected graph. Then the smallest $k \in \mathbb{N}$ for which (G, k) is a YES instance of ML is:*

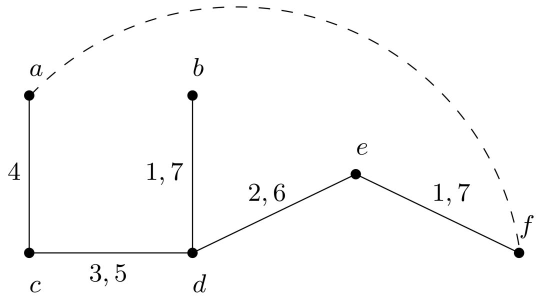
$$k = \begin{cases} 2n - 4, & \textit{if } G \textit{ contains } C_4 \textit{ as a subgraph,} \\ 2n - 3, & \textit{otherwise.} \end{cases}$$

In a C_4-free graph with n vertices, an optimal solution to ML consists in assigning in total $2n - 3$ time-labels to the $n - 1$ edges of a spanning tree. In such a solution, one of these $n - 1$ edges receives one time-label, while each of the remaining $n - 2$ edges receives two time-labels. Similarly, when the graph contains a C_4, it suffices to span the graph with four trees rooted at the vertices of the C_4, where each of the edges of the C_4 receives one time-label and each edge of the four trees receives two labels. That is, a graph containing a C_4 can be temporally connected using $2n - 4$ time-labels. An example of a labeling achieving the bounds is presented in Figure 5.1.

As a second warm-up, we show that the minimization analogues of ML and MAL on directed acyclic graphs (DAGs) are solvable in polynomial time. More specifically, for the minimization analogue of ML we provide an algorithm which, given a DAG $G = (V, A)$ with diameter d_G, computes a temporal labeling function λ which assigns the smallest possible number of time-labels on the arcs of G. We do this by first computing the Transitive reduction G' of G (introduced by Aho et al. [2]), which is a subgraph of G with the same vertex set and the following property:



(a) An example of a labeling that temporally connects a graph that contains a C_4, where the C_4 edges receive one label and all other edges of a spanning tree receive two labels.



(b) An example of a labeling that temporally connects a graph that does not contain a C_4, where one edge receives one label and all other edges of a spanning tree receive two labels.

Figure 5.1: An example of labeling meeting bounds from Theorem 5.2.5 for a graph containing a C_4 (Figure 5.1a) and a graph without a C_4 (Figure 5.1b). We mark the edges of a spanning tree or a spanning tree with a C_4 with a solid line and all other edges with a dashed line.

for every two vertices $u, v \in V(G)$, there exists a directed path from u to v in G' if and only if there exists a directed path from u to v in G. We then assign one label to each edge $E(G') \subseteq E(G)$ which results in the existence of a directed temporal path in (G, λ) from a vertex u to v, whenever there is a directed path from u to v in G. Moreover, the age $\alpha(G, \lambda)$ of the resulting temporal graph is *equal* to d_G. Therefore, this immediately implies a polynomial-time algorithm for the minimization analogue of MAL on DAGs. As a contrast, Mertzios et al. [98] proved that MAL is APX-hard on general directed graphs. The more relaxed version of the problem on directed graphs, where the age is unbounded, still remains open.

Theorem 5.2.6. *Let $G = (V, E)$ be a DAG with n vertices and m arcs. Then $ML(G)$ and $MAL(G)$ can be both solved in polynomial time.*

Proof. We have to specify labeling λ of G that would make (G, λ) temporally connected while assigning the least number of labels to the edges of G. We start by creating the transitive reduction G' of G. We then collect all source vertices of G' to a set S_1 and assign the label 1 to all arcs in (G, λ) incident to the vertices in S_1. We remove all vertices of S_1 from G' together with all of their incident arcs and repeat the above while increasing the label for 1. We proceed until there are no more arcs in G'. At the end (G, λ) is a temporally connected graph. Moreover, the largest label used is the diameter d_G of G, which follows directly from the definition of the transitive reduction. If the input k is smaller than $|\lambda|$, or if the input age restriction a is smaller than the diameter of G, our algorithm returns No, else returns Yes.

It is known that the transitive reduction is constructed in polynomial time (see [2]). All other operations need polynomial time too, therefore the whole algorithm is executed in polynomial time.

Let us now prove the correctness of our procedure. By the definition $E' = E(G')$ is the minimum collection of arcs of G that preserve the reachability of G. This means that removing any arc from E' disconnects the graph. Therefore, λ has to assign at least one label to each arc from E' in order for (G, λ) to be temporally connected. Our algorithm assigns exactly one label to each arc in E', therefore $|\lambda|$ is of the smallest possible size. Now, to show that (G, λ) is temporally connected, let u, v be two arbitrary vertices in G, where there exists a directed path from u to v in G. We will show that there is also a directed temporal path from u to v in the resulting temporal graph (G, λ). Since there is a directed path from u to v in G, there is also a directed path from u to v in G'. Denote it as $P = (u = w_0, w_1, \ldots, w_k = v)$, where $k \geq 1$. Let us observe any two consecutive arcs (w_{i-1}, w_i) and (w_i, w_{i+1}) from the path P. Our algorithm will label the first arc (w_{i-1}, w_i) with a label that is strictly smaller than the label of the second arc (w_i, w_{i+1}). This is true as the second arc gets labeled, when w_i will be a source vertex in the modified G' and all of its incoming arcs, including (w_{i-1}, w_i), are labeled and removed from G'. Therefore, (P, λ) temporally connects the vertex u to vertex v. $\square$

5.3 MAL is NP-complete

In this section, we prove that it is NP-hard to determine the number of labels in an optimal labeling of a static, undirected graph G, where the age, i. e., the maximum label used, is equal to the diameter d of the input graph. It is worth noting here that, for any $x \geq 1$, the complexity of MAL remains open in the case where the age is allowed to be at most $d + x$. For a full picture of problem complexity, we would like to remind the reader that the directed analogue of the problem (the case when the input graph G is a directed graph), was shown to be APX-hard, even when the maximum length of a directed cycle is 2, see [98], while we show that the problem is polynomial-time solvable for DAGs, see Theorem 5.2.6.

To prove the NP-hardness we provide a reduction from the problem Monotone Max XOR(3) (or MonMaxXOR(3) for short). This is a special case of the classical Boolean satisfiability problem, where the input formula ϕ consists of the conjunction of *monotone* XOR clauses of the form $(x_i \oplus x_j)$, i. e., variables x_i, x_j are non-negated. If each variable appears in exactly r clauses, then ϕ is called a *monotone* Max XOR(r) formula. A clause $(x_i \oplus x_j)$ is *XOR-satisfied* (or simply *satisfied*) if and only if $x_i \neq x_j$. In Monotone Max XOR(r) we are trying to compute a truth assignment τ of ϕ which satisfies the greatest possible number of clauses.

Max-Cut on cubic graphs reduces to MonMaxXOR(3) using the following reduction. Given a cubic graph G for each vertex $v \in V(G)$ create a variable x_v in the MonMaxXOR(3) formula ϕ_G. For every edge $uv \in E(G)$, add the clause $(x_v \oplus x_u)$ to ϕ_G. It is easy to see that computing a maximum cut in G (i. e., a partition of $V(G)$ into two sets A and $\overline{A}$ such that the number $|\{uv \in E(G) : u \in A, v \in \overline{A}\}|$ of edges between A and $\overline{A}$ is maximized), is equivalent to computing a maximum number of satisfied clauses in ϕ_G. Since Max-Cut is known to be NP-hard [6], we conclude the following.

Theorem 5.3.1 ([6]). *MonMaxXOR(3) is NP-hard.*

We now describe our reduction from MonMaxXOR(3) to the problem Minimum Aged Labeling (MAL), where the input static graph G is undirected and the desired age of the output temporal graph is the diameter d of G. Let ϕ be a monotone Max XOR(3) formula with n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$. Note that $m = \frac{3}{2}n$, since each variable appears in exactly 3 clauses. From ϕ we construct a static undirected graph G_ϕ with diameter $d_\phi = 10$, and prove that there exists a truth assignment τ which satisfies at least k clauses in ϕ, if and only if there exists a labeling λ_ϕ of G_ϕ, with $|\lambda_\phi| \leq 7n^2 + 49n - 8k$ labels, where the maximum used label is d_ϕ.

High-level construction For each variable x_i, $1 \leq i \leq n$, we construct a variable gadget X_i that consists of a "starting" vertex s_i and three "ending" vertices t_i^ℓ (for $\ell \in \{1,2,3\}$); these ending vertices correspond to the appearances of x_i in three clauses of ϕ. In an optimum labeling λ_ϕ, in each variable gadget, there are exactly two labelings that temporally connect starting and ending vertices, which corresponds to the TRUE or FALSE truth assignment of the variable in the input formula ϕ. For every clause $(x_i \oplus x_j)$ we identify corresponding ending vertices of X_i and X_j (as well as some other auxiliary vertices and edges). Whenever $(x_i \oplus x_j)$ is satisfied by a truth assignment of ϕ, the labels of the common edges of X_i and X_j in an optimum labeling coincide (thus using few labels); otherwise, we need additional labels for the common edges of X_i and X_j.

Detailed construction of G_ϕ For each variable x_i from ϕ we create a variable gadget X_i (for an illustration see Figure 5.2), that consists of a *base* BX_i on 11 vertices, $BX_i = \{s_i, a_i, b_i, c_i, d_i, e_i, \overline{a_i}, \overline{b_i}, \overline{c_i}, \overline{d_i}, \overline{e_i}\}$, and three *forks* F^1X_i, F^2X_i, F^3X_i, each on 9 vertices, $F^\ell X_i = \{t_i^\ell, f_i^\ell, g_i^\ell, h_i^\ell, m_i^\ell, \overline{f_i}^\ell, \overline{g_i}^\ell, \overline{h_i}^\ell, \overline{m_i}^\ell\}$, where $\ell \in \{1,2,3\}$. Vertices in the base BX_i are connected in the following way: there are two paths of length 5: $s_i a_i b_i c_i d_i e_i$ and $s_i \overline{a_i}\,\overline{b_i}\,\overline{c_i}\,\overline{d_i}\,\overline{e_i}$, and 5 extra edges of form $y_i\overline{y_i}$, where $y \in \{a,b,c,d,e\}$. Vertices in each fork $F^\ell X_i$ (where $\ell \in \{1,2,3\}$) are connected in the following way: there are two paths of length 4: $t_i^\ell m_i^\ell h_i^\ell g_i^\ell f_i^\ell$ and $t_i^\ell \overline{m_i}^\ell \overline{h_i}^\ell \overline{g_i}^\ell \overline{f_i}^\ell$, and 4 extra edges of form $y_i\overline{y_i}^\ell$, where $y \in \{m,h,g,f\}$. The base BX_i of the variable gadget X_i is connected to each of the three forks $F^\ell X_i$ via two edges $e_i f_i^\ell$ and $\overline{e_i}\,\overline{f_i}^\ell$, where $\ell \in \{1,2,3\}$.

For an easier analysis, we fix the following notation. The vertex $s_i \in BX_i$ is called a *starting vertex* of X_i, vertices t_i^ℓ ($\ell \in \{1,2,3\}$) are called *ending vertices* of X_i. Vertices $a_i, b_i, c_i, d_i, e_i, f_i^\ell, g_i^\ell, h_i^\ell, m_i^\ell$ (resp. $\overline{a_i}, \overline{b_i}, \ldots \overline{m_i}^\ell$) are called the left (resp. the right) vertices of X_i. A path connecting s_i, t_i^ℓ that passes only through the left (resp. the right) vertices is called the *left* (resp. *right*) s_i, t_i^ℓ-path. The left (resp. right) s_i, t_i^ℓ-path is a disjoint union of the left (resp. right) path on vertices of the base BX_i of X_i, an edge of form $e_i f_i^\ell$ (resp. $\overline{e_i}\,\overline{f_i}^\ell$) called the left (resp. right) *bridge edge* and the left (resp. right) path on vertices of the ℓ-th fork $F^\ell X_i$ of X_i. The edges $y_i\overline{y_i}$, where $y \in \{a, b, c, d, e, f^\ell, g^\ell, h^\ell, m^\ell\}$, $\ell \in \{1, 2, 3\}$, are called *connecting edges*.

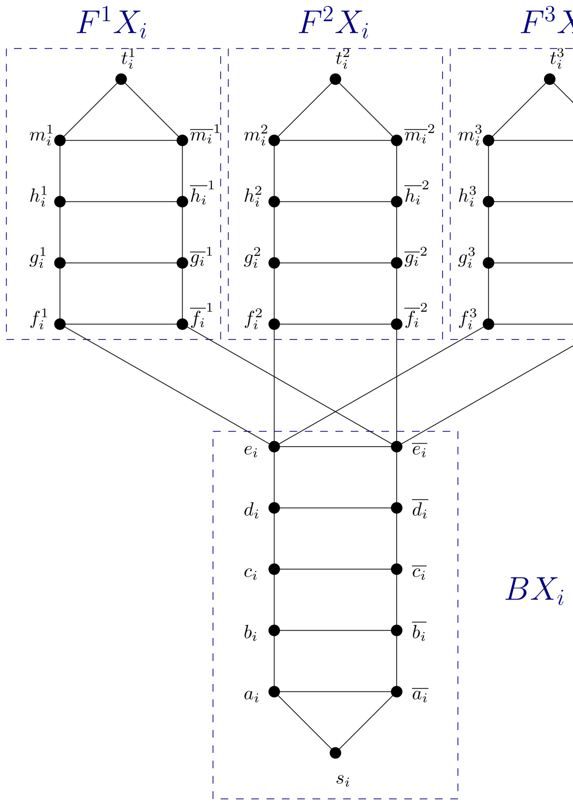


Figure 5.2: An example of a variable gadget X_i in G_ϕ, corresponding to the variable x_i from ϕ.

Connecting variable gadgets There are two ways in which we connect two variable gadgets, depending on whether they appear in the same clause in ϕ or not.

1. Two variables x_i, x_j do not appear in any clause together (for an illustration see Figure 5.3). In this case we add the following edges between the variable gadgets X_i and X_j:

    - from e_i (resp. $\overline{e_i}$) to $f_j^{\ell'}$ and $\overline{f_j}^{\ell'}$, where $\ell' \in \{1, 2, 3\}$,
    - from e_j (resp. $\overline{e_j}$) to f_i^{ℓ} and $\overline{f_i}^{\ell}$, where $\ell \in \{1, 2, 3\}$,
    - from d_i (resp. $\overline{d_i}$) to d_j and $\overline{d_j}$.

    We call these edges the *inter-variable edges*.

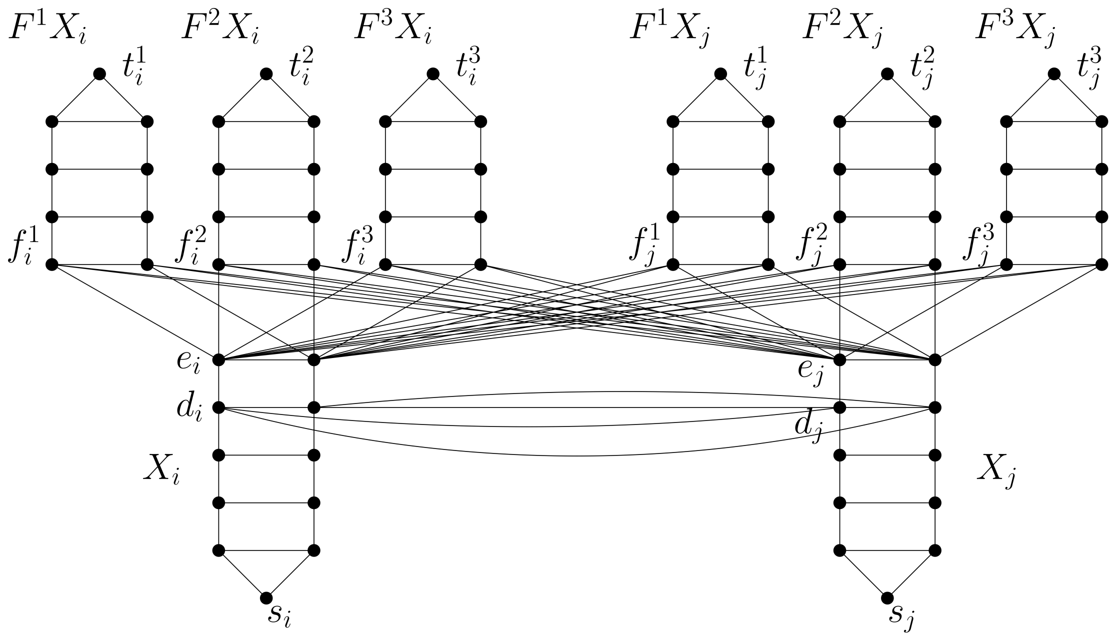


Figure 5.3: An example of two non-intersecting variable gadgets and inter-variable edges among them.

2. Two variables appear in a clause together (for an illustration see Figure 5.4). Let $C = (x_i \oplus x_j)$ be a clause of ϕ, that contains the r-th appearance of the variable x_i and r'-th appearance of the variable x_j. In this case we identify the r-th fork $F^r X_i$ of X_i with the r'-th fork $F^{r'} X_j$ of X_j in the following way:

   - $t_i^r = t_j^{r'}$,
   - $\{f_i^r, g_i^r, h_i^r, m_i^r\} = \{\overline{f_j}^{r'}, \overline{g_j}^{r'}, \overline{h_j}^{r'}, \overline{m_j}^{r'}\}$ respectively, and
   - $\{\overline{f_i}^r, \overline{g_i}^r, \overline{h_i}^r, \overline{m_i}^r\} = \{f_j^{r'}, g_j^{r'}, h_j^{r'}, m_j^{r'}\}$ respectively.

   Besides that, we add the following edges between the variable gadgets X_i and X_j:

   - from e_i (resp. $\overline{e_i}$) to $f_j^{\ell'}$ and $\overline{f_j}^{\ell'}$, where $\ell' \in \{1, 2, 3\} \setminus \{r'\}$,
   - from e_j (resp. $\overline{e_j}$) to f_i^{ℓ} and $\overline{f_i}^{\ell}$, where $\ell \in \{1, 2, 3\} \setminus \{r\}$,
   - from d_i (resp. $\overline{d_i}$) to d_j and $\overline{d_j}$.

This finishes the construction of G_ϕ. Before continuing with the reduction, we prove the following structural property of G_ϕ.

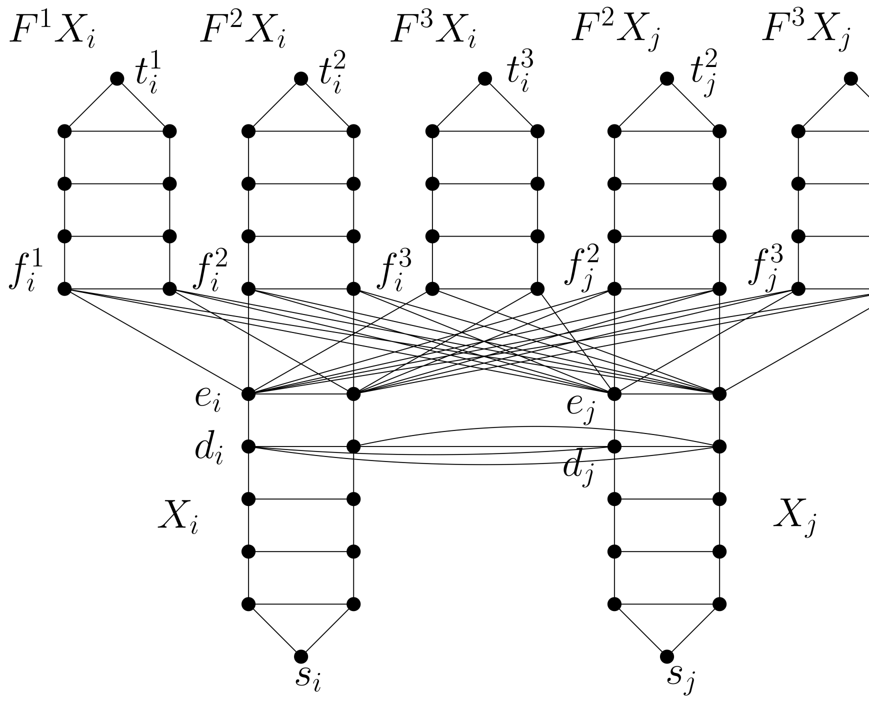


Figure 5.4: An example of two intersecting variable gadgets X_i, X_j corresponding to variables x_i, x_j, that appear together in some clause in ϕ, where it is the third appearance of x_i and the first appearance of x_j.

Lemma 5.3.2. *The diameter d_ϕ of G_ϕ is* 10.

Proof. We prove this in two steps. First, we show that the diameter of any variable gadget is 10 and then show that there exists a path of length at most 10 between any two vertices from two different variable gadgets, which proves the desired bound.

Let us start with fixing a variable gadget X_i. A path from the starting vertex s_i to any ending vertex t_i^ℓ ($\ell \in \{1,2,3\}$) has to go through at least one of the vertices from each of the following sets $\{a_i, \overline{a_i}\}, \{b_i, \overline{b_i}\}, \{c_i, \overline{c_i}\}, \{d_i, \overline{d_i}\}, \{e_i, \overline{e_i}\}, \{f_i^\ell, \overline{f_i}^\ell\}, \{g_i^\ell, \overline{g_i}^\ell\}$, $\{h_i^\ell, \overline{h_i}^\ell\}, \{m_i^\ell, \overline{m_i}^\ell\}$, before reaching the ending vertex. The shortest s_i, t_i^ℓ path will go through exactly one vertex from each of the above sets, therefore it is of length 10. A path between any two ending vertices $t_i^{\ell_1}, t_i^{\ell_2}$ (where $\ell_1, \ell_2 \in \{1,2,3\}$ and $\ell_1 \neq \ell_2$), has to go through at least one of the vertices from each of the following sets $\{m_i^{\ell_1}, \overline{m_i}^{\ell_1}\}$,$\{m_i^{\ell_2}, \overline{m_i}^{\ell_2}\}$, $\{h_i^{\ell_1}, \overline{h_i}^{\ell_1}\}$,$\{h_i^{\ell_2}, \overline{h_i}^{\ell_2}\}$, $\{g_i^{\ell_1}, \overline{g_i}^{\ell_1}\}$,$\{g_i^{\ell_2}, \overline{g_i}^{\ell_2}\}$, $\{f_i^{\ell_1}, \overline{f_i}^{\ell_1}\}$, $\{f_i^{\ell_2}, \overline{f_i}^{\ell_2}\}, \{e_i, \overline{e_i}\}$. Similarly, as before, the shortest path uses exactly one vertex from each set and is of size 10. It is not hard to see that the distance between any other vertex $v \in X_i \setminus \{s_i, t_i^\ell\}$ (where $\ell \in \{1,2,3\}$) and the starting vertex or one of the ending vertices is at most 9, as vertex v lies on one of the shortest (s_i, t_i^ℓ) or

$(t_i^{\ell_1}, t_i^{\ell_2})$ paths (where $\ell_1, \ell_2 \in \{1,2,3\}$ and $\ell_1 \neq \ell_2$), but it is not an endpoint of it. By similar reasoning there exists a path between any two vertices $u, v \in X_i \setminus \{s_i, t_i^\ell\}$ (where $\ell \in \{1,2,3\}$), of distance at most 9. Therefore, the diameter of X_i is 10.

Now we want to show that the distance between any two vertices from different variable gadgets is at most 10. Let us start with the case where two variable gadgets X_i and X_j share no fork (i. e., x_i and x_j do not appear in the same clause of ϕ). A path between s_i and t_j^ℓ (for $\ell \in \{1,2,3\}$) travels through at least one of the vertices from the following sets $\{a_i, \overline{a_i}\}, \{b_i, \overline{b_i}\}, \ldots, \{e_i, \overline{e_i}\}, \{f_j^\ell, \overline{f_j}^\ell\}, \{g_j^\ell, \overline{g_j}^\ell\}, \ldots, \{m_j^\ell, \overline{m_j}^\ell\}$. The shortest path goes through exactly one vertex in each of the sets, therefore it is of length 10. From this, it also follows that there exists a path between any base vertex $v \in BX_i$ and fork vertex $u \in FX_j$ of length at most 10. Next, observe a path between s_i and s_j that goes through at least one of the vertices from each of the following sets $\{a_i, \overline{a_i}\}, \{b_i, \overline{b_i}\}, \{c_i, \overline{c_i}\}, \{d_i, \overline{d_i}\}, \{d_j, \overline{d_j}\}, \{c_j, \overline{c_j}\}, \{b_j, \overline{b_j}\}, \{a_j, \overline{a_j}\}$. Again, the shortest path will use exactly one vertex in each set and is of distance 9. Therefore, all of the a, b, c, d vertices from X_i and X_j are on the distance at most 9 from each other. Since the path (e_i, f_j^1, e_j) is of length 2 and $(e_i, d_i, d_j, c_j, b_j, a_j, s_j)$ is of length 6 it follows that e_i is at distance at most 3 from e_j and 6 from s_j. Therefore, all of the vertices from BX_i and BX_j are on the distance at most 9 from each other. Lastly, a path between $t_i^{\ell_1}$ and $t_j^{\ell_2}$ (where $\ell_1, \ell_2 \in \{1,2,3\}$) travels through at least one of the vertices from the following sets $\{m_i^{\ell_1}, \overline{m_i}^{\ell_1}\}, \{h_i^{\ell_1}, \overline{h_i}^{\ell_1}\}, \{g_i^{\ell_1}, \overline{g_i}^{\ell_1}\}, \{f_i^{\ell_1}, \overline{f_i}^{\ell_1}\}, \{e_i, \overline{e_i}\}, \{f_j^{\ell_2}, \overline{f_j}^{\ell_2}\}, \{g_j^{\ell_2}, \overline{g_j}^{\ell_2}\}, \{h_j^{\ell_2}, \overline{h_j}^{\ell_2}\}$, $\{m_j^{\ell_2}, \overline{m_j}^{\ell_2}\}$. Since the shortest path visits exactly one vertex from each set, it is of length 10. Similarly, as before, it follows that there is a path between any two vertices $u \in F^{\ell_1}X_i$ and $v \in F^{\ell_2}X_j$ (where $\ell_1, \ell_2 \in \{1,2,3\}$) of distance at most 10. Therefore, we get that the diameter of a subgraph of G_ϕ that contains any two variable gadgets that do not share a fork is 10. In the case when two variable gadgets X_i and X_j share a fork, it is not hard to see that the shortest path among any two vertices $u \in X_i$ and $v \in X_j$ does not become greater than in the case when two variable gadgets do not share a fork.

Altogether it follows that the distance among any two vertices in G_ϕ is at most 10. $\square$

Lemma 5.3.3. *If $OPT_{\textsc{MonMaxXOR(3)}}(\phi) \geq k$ then $OPT_{MAL}(G_\phi, d_\phi) \leq 7n^2+49n-8k$, where n is the number of variables in the formula ϕ.*

Proof. Let τ be an optimum truth assignment of ϕ, i. e., a truth assignment that satisfies at least k clauses of ϕ. We will prove that there exists a temporal labeling λ_ϕ of G_ϕ which uses $|\lambda_\phi| \leq 7n^2 + 49n - 8k$ labels, such that (G_ϕ, λ_ϕ) is temporally connected and $\alpha(G_\phi, \lambda_\phi) = d_\phi = 10$. Recall that since ϕ is an instance of MonMaxXOR(3) with n variables it has $m = \frac{3}{2}n$ clauses. We build the labeling λ_ϕ using the following rules. For an illustration see Figure 5.5.

1. If a variable x_i from ϕ is set to True by the truth assignment τ, we label the edges in X_i in the following way:

   - all three left (s_i, t_i^ℓ)-paths, for all $\ell \in \{1, 2, 3\}$, get the labels $1, 2, 3, \dots, 10$, one on each edge,
   - similarly, all left (t_i^ℓ, s_i)-paths, get the labels $1, 2, 3, \dots, 10$, one on each edge,
   - all connecting edges (i. e., edges of form $y_i\overline{y_i}$, where $y \in \{a, b, c, d, e, f^\ell, g^\ell, h^\ell, m^\ell\}$) get the labels 1 and 10.

   If a variable x_i from ϕ is set to False by the truth assignment τ, we label the edges in X_i in the following way:

   - all three right (s_i, t_i^ℓ)-paths, for all $\ell \in \{1, 2, 3\}$, get the labels $1, 2, 3, \dots, 10$, one on each edge,
   - similarly, all right (t_i^ℓ, s_i)-paths, get the labels $1, 2, 3, \dots, 10$, one on each edge,
   - all connecting edges get the labels 1 and 10.

   Labeling λ_ϕ uses 10 labels on the left (resp. right) path of the base BX_i, 10 labels on the left (resp. right) path of each fork $F^\ell X_i$, where $\ell \in \{1, 2, 3\}$ and $10 + 3 \cdot 8$ labels on the connecting edges. All in total λ_ϕ uses 74 labels on the variable gadget X_i.

We now need to prove that there exists a temporal path among any two vertices in X_i. Suppose x_i is set to True in the truth assignment τ of ϕ (the case of x_i being False is analogous). By the construction of λ_ϕ, there are temporal paths from s_i to any of the t_i^ℓ, where $\ell \in \{1, 2, 3\}$, and vice versa. Labeling λ_ϕ of G_ϕ gives rise to the following temporal paths. There is a temporal path from the starting vertex s_i to the ending vertex t_i^ℓ, where $\ell \in \{1, 2, 3\}$, which uses the left path of X_i, and labels $1, 2, \ldots, 10$. Similarly it holds for the temporal (t_i^ℓ, s_i)-path. The temporal path connecting two ending vertices $t_i^{\ell_1}, t_i^{\ell_2}$ (where $\ell_1, \ell_2 \in \{1, 2, 3\}$ and $\ell_1 \neq \ell_2$), uses first the left path of the fork $F^{\ell_1}X_i$, with labels 1 to 5, to reach e_i, and then continues on the left path of the fork $F^{\ell_2}X_i$ using labels 6 to 10. Since the temporal paths among starting and ending vertices use the left path of the gadget X_i it follows that all vertices on the left path reach all starting and ending vertices, and consequently, they also reach each other. Any remaining vertex, i. e., a vertex on the right path of the gadget X_i, can reach the starting vertex using first their corresponding connecting edge at time 1, and then the remaining part of the temporal path from t_i^ℓ (for $\ell \in \{1, 2, 3\}$) to s_i. Similarly, it holds for the temporal paths towards all of the ending vertices. In the case of temporal paths from s_i (or t_i^ℓ) to the vertices on the right side of X_i, the temporal paths start with the edges of the left path of X_i at time 1 and finish using the corresponding connecting edge at time 10. Lastly, temporal paths among two vertices from the right path of X_i use as a first and last edge the corresponding connecting edge at time 1 and 10 respectively, and a part of the (s_i, t_i^ℓ) or (t_i^ℓ, s_i)-temporal path. This proves that the labeling λ_ϕ of X_i admits a temporal path among any two vertices in X_i.

2. If two variable gadgets X_i and X_j do not share a fork, i. e., variables x_i and x_j are not in the same clause in ϕ, and are both set to True by τ, then we label the inter-variable edges as follows:

   - the edge $d_i d_j$, connecting the left path of BX_i with the left path of BX_j, gets labels 5 and 6,

- three edges of the form $e_i f_j^{\ell'}$ ($\ell' \in \{1,2,3\}$) that connect the left path of BX_i to the left paths of $F^{\ell'}X_j$ get labels 5 and 6,
- three edges of the form $e_j f_i^{\ell}$ ($\ell \in \{1,2,3\}$) that connect the left path of BX_j to the left paths of $F^{\ell}X_i$ get labels 5 and 6.

We have assigned 14 labels to 7 inter-variable edges that connect both variable gadgets, while the number of labels assigned to each variable gadget remains the same. Note that the other three combinations (x_i, x_j are both FALSE, one of x_i, x_j is TRUE and the other FALSE) give rise to the labeling λ_ϕ that uses the same number of labels on both variable gadgets and inter-variable edges, where the labeled inter-variable edges are chosen appropriately. For an example see Figure 5.5a.

Since labeling inter-variable edges does not change the labeling on each variable gadget, we know that there is still a temporal path among any two vertices from the same variable gadget. We need to prove now that there is a temporal path among any two vertices from X_i and X_j. First observe that there is a unique temporal path from s_i to t_j^{ℓ} (for $\ell \in \{1,2,3\}$), that first uses the left path of the base of X_i with labels $1,2,3,4,5$, the inter-variable edge $e_i f_j^{\ell}$ with label 6 and continues to t_j^{ℓ} using the left path of the fork F_j^{ℓ} with labels $7,8,9,10$. The reverse (t_j^{ℓ}, s_i)-temporal path uses the same edges with labels $1,2,\ldots,10$, as defined by λ_ϕ. From the above, it follows that there exists a temporal path from any vertex in the base of BX_i to any vertex in a fork F_j^{ℓ} and vice versa (note, if any of the starting/ending vertices are not on a left path of X_i or X_j we use corresponding connecting edges at time 1 or 10). Next, we show that there is a temporal path between two ending vertices $t_i^{\ell_1} \in X_i$ and $t_j^{\ell_2} \in X_j$ (where $\ell_1, \ell_2 \in \{1,2,3\}$). More specifically, the $(t_i^{\ell_1}, t_j^{\ell_2})$-temporal path first uses the left side of the fork $F_i^{\ell_1}$ with labels $1,2,3,4,5$ to reach the vertex $e_i \in X_i$ and then uses the bridge edge $e_i f_j^{\ell_2}$ at time 6 and continues on the left side of the fork $F_j^{\ell_2}$ with labels $7,8,9,10$ to reach $t_j^{\ell_2}$. Thus it holds that any vertex in any of the forks of X_i can reach any vertex in any of the forks of X_j. Note that the last temporal path proves also that $e_i \in X_i$ reaches all of the vertices in all of the forks of X_j (and vice versa). Let us now show that e_i reaches also all the vertices in the base of X_j (and vice versa). First, the (e_i, e_j)-temporal path is of length 2, starts with the bridge edge $e_i f_j^{\ell_2}$ at time 5 and finishes with the edge $f_j^{\ell_2} e_j$ at time 6. Second, e_i reaches vertex s_j using the temporal path that travels through vertices $e_i, d_i, d_j, c_j b_j, a_j, s_j$ with labels $5, 6, 7, 8, 9, 10$ on the respective edges. Conversely, the (s_j, e_i)-temporal path travels through the same vertices $s_j, a_j, b_j, c_j, d_j, d_i, e_i$ with labels $1, 2, 3, 4, 5, 6$ on the respective edges. From the above three temporal paths, it follows that e_i in fact does temporally reach all of the vertices in the base of X_j and vice versa. Lastly, we want to prove that all of the remaining base vertices of X_i (i.e. vertices of form a, b, c, d) reach all of the remaining base vertices in X_j. To do so we just have to provide a temporal path from s_i to s_j. This temporal path travels through the vertices $s_i, a_i, b_i, c_i, d_i, d_j, c_j, b_j, a_j, s_j$ using labels $1, 2, 3, 4, 5, 7, 8, 9, 10$ on the respective edges. All of the above proves that there exists a temporal path among any two vertices in X_i and X_j, when X_i and X_j share no fork.

3. If two variable gadgets X_i and X_j share a fork, i. e., variables x_i and x_j are in the same clause, are both set to TRUE and $F^r X_i = F^{r'} X_j$, then we label the following inter-variable edges:

   - the edge $d_i d_j$ connecting the left path of BX_i and BX_j gets labels 5 and 6,
   - two edges of the form $e_i f_j^{\ell'}$ ($\ell' \in \{1, 2, 3\} \setminus \{r'\}$) that connect the left path of BX_i to the left paths of $F^{\ell'} X_j$ get labels 5 and 6,
   - two edges of the form $e_j f_i^{\ell}$ ($\ell \in \{1, 2, 3\} \setminus \{r\}$) that connect the left path of BX_j to the left paths of $F^{\ell} X_i$ get labels 5 and 6.

We have assigned 10 labels to 5 inter-variable edges that connect both variable gadgets. Note that the three other combinations (x_i, x_j are both FALSE, one of x_i, x_j is TRUE and the other FALSE) give rise to the labeling λ_ϕ that uses the same number of labels on inter-variable edges, where the labeled edges are chosen according to the truth values of x_i and x_j. The only difference is in the labeling of the shared fork $F^r X_i = F^{r'} X_j$. There are two possibilities, one when the truth value of x_i and x_j is the same and one when it is different, i. e., $x_i = x_j$ or $x_i \neq x_j$.

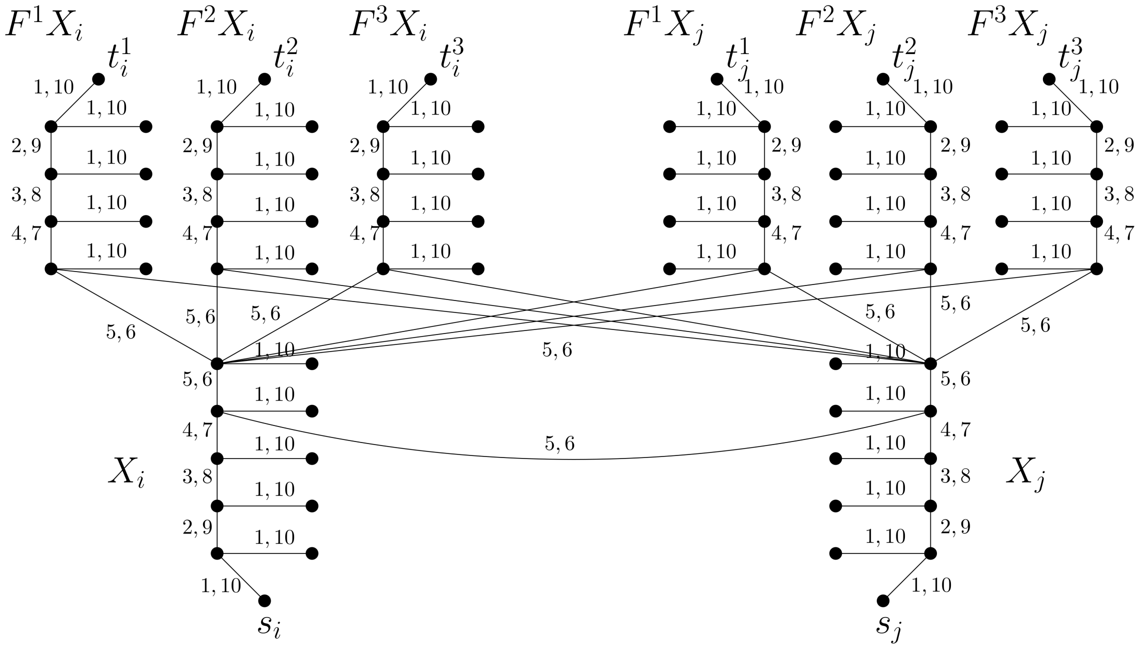
(a) Let us start with the case when $x_i \neq x_j$. Then the labeling λ_ϕ of $F^r X_i$ coincides with the labeling of $F^{r'} X_j$. Therefore λ_ϕ uses 16 less labels on the shared fork.

(b) In the case when $x_i = x_j$. The fork $F^r X_i = F^{r'} X_j$ gets labeled from both sides, i. e., all edges in the fork get 2 labels. Therefore λ_ϕ uses 8 less labels on the shared fork.

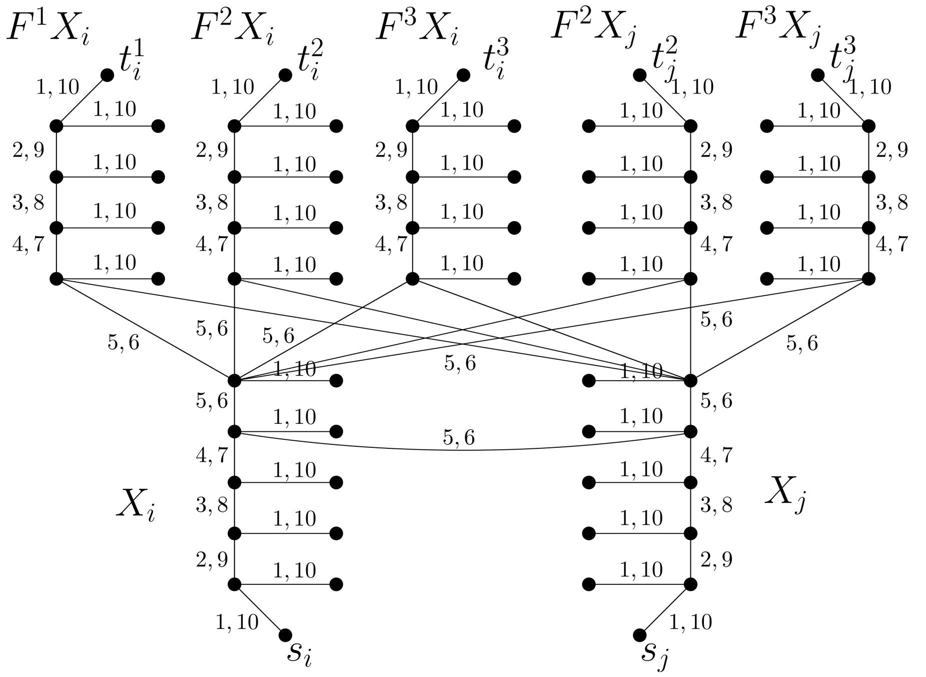
Identifying two forks $F^r X_i = F^{r'} X_j$ and labeling them using the union of both labelings on each fork, clearly preserves temporal paths among all the vertices from X_i (resp. X_j). What remains to check is that all vertices in X_i reach all the vertices in X_j. This follows from the same proof as in the previous case, where the paths between the two variable gadgets use the appropriate inter-variable edges. Note, since the fork $F^r X_i = F^{r'} X_j$ is in the intersection, the inter-variable edges from X_i (resp. X_j) to $F^r X_i = F^{r'} X_j$ do not exist. Therefore, the labeling λ_ϕ admits a temporal path among any two vertices from the variable gadgets X_i, X_j, that have a fork in the intersection.

Summarizing all of the above we get that the labeling λ_ϕ uses 74 labels on each variable gadget and 14 labels on inter-variable edges among any two variables, from which we have to subtract the following:

- 4 labels for each pair of inter-variable edges between two variables that appear in the same clause,
- 16 labels for the shared fork between two variables, that appear in a satisfied clause,
- 8 labels for the shared fork between two variables, that appear in a non-satisfied clause.



(a) x_i and x_j do not appear together in any clause.



(b) x_i and x_j appear together in a clause, where x_i appears with its third and x_j with its first appearance.

Figure 5.5: Example of the labeling λ on variable gadgets X_i, X_j and inter-variable edges between them, where x_i is TRUE and x_j FALSE in ϕ. Note that edges that are not labeled are omitted, $F^3X_i = F^1X_j$ and $t_i^3 = t_j^1$.

Altogether this sums up to $74n + 14\frac{n(n-1)}{2} - 4m - 16k - 8(m-k)$. As a result, given that τ satisfies a minimum of k clauses of ϕ, the labeling λ_ϕ admits at most $7n^2 + 49n - 8k$ labels. $\square$

Before proving the statement in the other direction, we have to show some structural properties. Let us fix the following notation. Every temporal path from s_i to t_i^ℓ (resp. from t_i^ℓ to s_i) of length 10 in X_i is called an *upward path* (resp. a *downward path*) in X_i. Any part of an upward (resp. downward) path is called a *partial* upward (resp. downward) path. Note that, for any $\ell, \ell' \in \{1,2,3\}$, $\ell \neq \ell'$, a temporal path from t_i^ℓ to $t_i^{\ell'}$ of length 10 is the union of a partial downward path on the fork F_i^ℓ and a partial upward path on $F_i^{\ell'}$. If a labeling λ_ϕ labels all left (resp. right) paths of the variable gadget X_i (i. e., both bottom-up from s_i to t_i^1, t_i^2, t_i^3 and top-down from t_i^1, t_i^2, t_i^3 to s_i with labels $1, 2 \ldots, 10$ in this order), then we say that the variable gadget X_i is *left-aligned* (resp. *right-aligned*) in the labeling λ_ϕ. Note that if at least one edge on any of these left (resp. right) paths of X_i is not labeled with the appropriate label between 1 and 10, then the variable gadget is *not* left-aligned (resp. *not* right-aligned). The following technical lemma will allow us to prove the correctness of our reduction.

Lemma 5.3.4. *Let λ_ϕ be a minimum labeling of G_ϕ. Then λ_ϕ can be modified in polynomial time to a minimum labeling of G_ϕ in which each variable gadget X_i is either left-aligned or right-aligned.*

Proof. Let λ_ϕ be a minimum labeling of G_ϕ that admits at least one variable gadget X_i that is neither left-aligned nor right-aligned (i.e. X_i does not admit all left upward and downward paths, or all right upward and downward paths).

First, we will prove that there exists a fork $F^\ell X_i$ of X_i that admits at least three partial upward or downward paths, i. e., it either has two partial upward paths (one on each side of the fork) and at least one partial downward path, or vice versa. For the sake of contradiction, suppose that each of the forks F^1X_i, F^2X_i, F^3X_i contains at most two partial paths. Then, since λ_ϕ must have in X_i at least one upward and at least one downward path between s_i and t_i^ℓ, $\ell \in \{1,2,3\}$, it follows that each fork $F^\ell X_i$ has *exactly* one partial upward and *exactly* one partial downward path.

Assume that each of the forks F^1X_i, F^2X_i, F^3X_i has both its partial upward and downward paths on the same side of X_i (i. e., either both on the left or both on the right side of X_i). If all of them have their partial upward and downward paths on the left (resp. right) side of X_i, then X_i is left-aligned (resp. right-aligned), which is a contradiction. Therefore, at least one fork (say F^1X_i) has its partial upward and downward paths on the left side of X_i and at least one other fork (say F^2X_i) has its partial upward and downward paths on the right side of X_i. But then there is no temporal path from t_i^1 to t_i^2 of length 10 in λ_ϕ, which is a contradiction. Therefore there exists at least one fork $F^\ell X_i$ (say, F^1X_i w.l.o.g.), in which (w.l.o.g.) the partial upward path is on the right side and the partial downward path is on the left side of X_i.

Since the partial downward path of F^1X_i is on the left side of X_i, it follows that the partial upward path of each of F^2X_i and F^3X_i is on the left side of X_i. Indeed, otherwise, there is no temporal path of length 10 from t_i^1 to t_i^2 or t_i^3 in λ_ϕ, a contradiction. Similarly, since the partial upward path of F^1X_i is on the right side of X_i, it follows that the partial downward path of each of F^2X_i and F^2X_i is on the right side of X_i. But then, there is no temporal path of length 10 from t_i^2 to t_i^3, or from t_i^3 to t_i^2 in λ_ϕ, which is also a contradiction. Therefore at least one fork $F^\ell X_i$ (say F^3X_i) of X_i admits at least three partial upward or downward paths.

W.l.o.g. we can assume that the fork F^3X_i has two partial downward paths and at least one partial upward path which is on the left side of X_i. We distinguish now the following cases.

Case A. The fork F^3X_i has no partial upward path on the right side of X_i. Then the base BX_i has a partial upward path on the left side of X_i. Furthermore, each of the forks F^1X_i, F^2X_i has a partial downward path on the left side of X_i. Indeed, if otherwise F^1X_i (resp. F^2X_i) has no partial downward path on the left side of X_i, then there is no path with at most 10 edges from t_i^1 (resp. t_i^2) to t_i^3, a contradiction.

Case A-1. The base BX_i of X_i has no partial downward path on the left side of X_i; that is, there is no temporal path from vertex e_i to vertex s_i with labels "6,7,8,9,10". Then the base BX_i of X_i has a partial downward path on the right side of X_i, as otherwise there would be no temporal path of length 10 from any of t_i^1, t_i^2, t_i^3 to s_i.

For the same reason, each of the forks F^1X_i, F^2X_i has a partial downward path on the right side of X_i.

Case A-1-i. None of the forks F^1X_i, F^2X_i has a partial upward path on the left side of X_i. Then each of the forks F^1X_i, F^2X_i has a partial upward path on the right side of X_i, as otherwise there would be no temporal path of length 10 from s_i to t_i^1 or t_i^2. For the same reason, the base BX_i has a partial upward path on the right side of X_i. Therefore we can remove the label "5" from the left bridge edge $e_if_i^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Case A-1-ii. Exactly one of the forks F^1X_i, F^2X_i (say F^1X_i) has a partial upward path on the left side of X_i. Then the fork F^2X_i has a partial upward path on the right side of X_i. Furthermore the base BX_i has a partial upward path on the right side of X_i, since otherwise there would be no temporal path of length 10 from s_i to t_i^2. In this case we can modify the solution as follows: remove the labels "1,2,3,4,5" from the partial right-upward path of BX_i and add the labels "6,7,8,9,10" to the partial left-upward path of the fork F^2X_i. Finally, we can remove the label "5" from the right bridge edge $\overline{e_if_i}^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Case A-1-iii. Each of the forks F^1X_i, F^2X_i has a partial upward path on the left side of X_i. In this case, we can modify the solution as follows: remove the labels "10,9,8,7,6" from the partial right-downward path of BX_i and add the same labels "10,9,8,7,6" to the partial left-downward path of the base BX_i. Finally, we can remove the label "5" from the right bridge edge $\overline{e_if_i}^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Case A-2. The base BX_i of X_i has a partial downward path on the left side of X_i; that is, there is a temporal path from vertex e_i to vertex s_i with labels "6,7,8,9,10".

Case A-2-i. None of the forks F^1X_i, F^2X_i has a partial upward path on the left side of X_i. Then the base BX_i and each of the forks F^1X_i, F^2X_i have a partial upward path on the right side of X_i, as otherwise there would be no temporal paths of length 10 from s_i to t_i^1, t_i^2. Moreover, as none of F^1X_i, F^2X_i has a partial left-upward path, it follows that each of F^1X_i, F^2X_i has a partial downward path on the right side of X_i. Indeed, otherwise, there would be no temporal paths of length 10 between t_i^1 and t_i^2. In this case, we can modify the solution as follows: remove the labels "1,2,3,4,5" from the partial left-upward path of BX_i and add the labels "6,7,8,9,10" to the partial right-upward path of the fork F^3X_i. Finally, we can remove the label "6" from the left bridge edge $e_if_i^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Case A-2-ii. Exactly one of the forks F^1X_i, F^2X_i (say F^1X_i) has a partial upward path on the right side of X_i. Then the fork F^2X_i has a partial upward path on the left side of X_i. Furthermore the base BX_i must have a partial right-upward path, as otherwise there would be no temporal path from s_i to t_i^2. In this case, we can modify the solution as follows: remove the labels "1,2,3,4,5" from the partial right-upward path of BX_i and add the labels "6,7,8,9,10" to the partial left-upward path of the fork F^2X_i. Finally, we can remove the label "5" from the right bridge edge $\overline{e_i}\overline{f_i}^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Case A-2-iii. Each of the forks F^1X_i, F^2X_i has a partial upward path on the right side of X_i. Then we we can simply remove the label "5" from the right bridge edge $\overline{e_i}\overline{f_i}^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Case B. The fork F^3X_i has also a partial upward path on the right side of X_i. That is, F^3X_i has partial upward-left, upward-right, downward-left, and downward-right paths.

Case B-1. The base BX_i of X_i has no partial downward path on the left side of X_i. Then the base BX_i of X_i has a partial downward path on the right side of X_i, as otherwise there would be no temporal path of length 10 from any of t_i^1, t_i^2, t_i^3 to s_i. For the same reason, each of the forks F^1X_i, F^2X_i has a partial downward path on the right side of X_i.

Note that Case B-1 is symmetric to the case where the base BX_i of X_i has no partial right-downward (resp. left-upward, right upward) path.

Case B-1-i. None of the forks F^1X_i, F^2X_i has a partial upward path on the left side of X_i. This case is the same as Case A-1-i.

Case B-1-ii. Exactly one of the forks F^1X_i, F^2X_i (say F^1X_i) has a partial upward path on the left side of X_i. Then both the base BX_i and the fork F^2X_i have a partial right-upward path, as otherwise there would be no temporal path of length 10 from s_i to t_i^2. In this case, we can always remove the label “6” from the left bridge edge $e_i f_i^3$ of the fork F^3X_i (without compromising the temporal connectivity), thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Case B-1-iii. Each of the forks F^1X_i, F^2X_i has a partial upward path on the left side of X_i. That is, each of F^1X_i, F^2X_i has a partial left-upward and a partial right-downward path. The following subcases can occur:

Case B-1-iii(a). None of the forks F^1X_i, F^2X_i has a partial right-upward path. Then each of the forks F^1X_i, F^2X_i has a partial left-downward path since otherwise there would not exist temporal paths of length 10 between t_i^1 and t_i^2. Furthermore, the base BX_i has a partial left-upward path, since otherwise there would not exist a temporal path of length 10 from s_i to t_i^1 and t_i^2. In this case, we can remove the label “6” from the right bridge edge $\overline{e_i f_i}^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Case B-1-iii(b). Exactly one of the forks F^1X_i, F^2X_i (say F^1X_i) has a partial right-upward path. Then the base BX_i has a partial left-upward path since otherwise there would not exist a temporal path of length 10 from s_i to t_i^2. Similarly, the fork F^1X_i has a partial left-downward path since otherwise there would not exist a temporal path of length 10 from t_i^1 to t_i^2. In this case, we can modify the solution as follows: First, remove the labels “10,9,8,7,6” from the partial right-downward path of BX_i and add the labels “10,9,8,7,6” to the partial left-downward path of BX_i. Second, remove the labels “5,6” from each of t two right bridge edges $\overline{e_i f_i}^1$ and $\overline{e_i f_i}^3$ of the forks F^1X_i and F^3X_i, respectively. Third, remove the label “5” from the right bridge edge $\overline{e_i f_i}^1$ of the fork F^2X_i. Finally, add the five labels “5,4,3,2,1” to the partial left-downward path of the fork F^2X_i. The resulting labeling λ_ϕ^* still preserves the temporal reachabilities and has the same number of labels as λ_ϕ, while the variable gadget X_i is aligned.

Case B-1-iii(c). Each of the forks F^1X_i, F^2X_i has a partial right-upward path. In this case, we can always remove the label “5” from the left bridge edge $e_i f_i^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Case B-2. The base BX_i of X_i has partial left-downward, right-downward, left-upward, and right-upward paths. Then, due to symmetry, we may assume w.l.o.g. that the fork F^1X_i has a left-upward path. Suppose that F^1X_i has also a left-downward path. In this case, we can modify the solution as follows: remove the labels "1,2,3,4,5" and "10,9,8,7,6" from the partial right-upward and right-downward paths of BX_i and add the labels "6,7,8,9,10" and "5,4,3,2,1" to the partial left-upward and left-downward paths of the fork F^2X_i. Finally, we can remove the label "6" from the right bridge edge $\overline{e_i}\overline{f_i}^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Finally, suppose that F^1X_i has no partial left-downward path. Then F^1X_i has a partial right-down path since otherwise there would not exist any temporal path of length 10 from t_i^1 to s_i. Similarly, the fork F^2X_i has a partial right-upward path since otherwise there would not exist any temporal path of length 10 from t_i^1 to t_i^2. In this case, we can modify the solution as follows: First, remove the labels "1,2,3,4,5" and "10,9,8,7,6" from the partial left-upward and left-downward paths of BX_i. Second, add the labels "6,7,8,9,10" to the partial right-upward path of the fork F^1X_i and add the labels "5,4,3,2,1" to the partial right-downward path of the fork F^2X_i. Finally remove the label "6" from the left bridge edge $e_if_i^3$ of the fork F^3X_i, thus obtaining a labeling with fewer labels than λ_ϕ, a contradiction.

Summarizing, starting from an optimum λ_ϕ of G_ϕ, in which at least one variable gadget is neither left-aligned nor right-aligned, we can modify λ_ϕ to another labeling λ^*_ϕ, such that λ^*_ϕ has one more variable-gadget that is aligned and $|\lambda_\phi| = |\lambda^*_\phi|$. Note that this modification can only happen in Case B-1-iii(b); in all other cases, our case analysis arrived at a contradiction. Note here that, by making the above modifications of λ_ϕ, we need to also appropriately modify the bridge edges (within the variable gadgets) and the inter-variable edges (between different variable gadgets), without changing the total number of labels in each of these edges. Finally, it is straightforward that all modifications of λ_ϕ can be done in polynomial time. This concludes the proof. □

Lemma 5.3.5. *If $OPT_{MAL}(G_\phi, d_\phi) \leq 7n^2+49n-8k$ then $OPT_{\text{MonMaxXOR(3)}}(\phi) \geq k$, where n is the number of variables in the formula ϕ.*

Proof. Recall that $d_\phi = 10$ (by Lemma 5.3.2). Let λ_ϕ be an optimum solution to $\text{MAL}(G_\phi, 10)$, which uses at most $7n^2 + 49n - 8k$ labels. We will prove that there exists a truth assignment τ that satisfies at least k clauses of ϕ. Lemma 5.3.4 implies that every variable gadget of G_ϕ is either left-aligned or right-aligned in λ_ϕ. Throughout the proof, we consider an arbitrary variable gadget X_i, and we assume w.l.o.g. that X_i is left-aligned.

First, we count the minimum number of labels needed in λ_ϕ, so that all temporal paths among vertices of X_i exist. Observe that s_i is at distance 10 from any t_i^ℓ, where $\ell \in \{1, 2, 3\}$, and that t_i^ℓ is at distance 10 from any $t_i^{\ell'}$, where $\ell' \in \{1, 2, 3\} \setminus \{\ell\}$. Therefore, any temporal path connecting any two of the vertices in $\{s_i, t_i^1, t_i^2, t_i^3\}$ is of length 10 and must use labels $1, 2, 3, \ldots, 10$ along its edges (in this order). As X_i is left-aligned, some of these temporal paths overlap, and thus there must exist at least $5 \cdot 2 = 10$ labels on the base BX_i, at least $4 \cdot 2 = 8$ labels on each fork $F^\ell X_i$, and at least 2 labels on each left-bridge edge $e_i f_i^\ell$. That is, we have at least $10 + 3 \cdot (8 + 2) = 40$ labels to temporally connect the vertices $\{s_i, t_i^1, t_i^2, t_i^3\}$ (and also all vertices within the paths among them at the left side of the variable gadget X_i). Furthermore, we need at least two labels for each vertex $\overline{y}$ at the right side of X_i such that there is a temporal path to and from $\overline{y}$ in X_i, i. e., we need at least $17 \cdot 2 = 34$ more labels in total. That is, within the whole variable gadget X_i we need in total at least $40 + 34 = 74$ labels in λ_ϕ.

Now, let X_j be a variable gadget in G_ϕ that does not share a fork with X_i. W.l.o.g. we can assume that X_j is right-aligned (all other cases are symmetric). As noted previously, temporal paths among vertices of one variable gadget do not use the inter-variable edges. Therefore, the inter-variable edges must be labeled in a way that ensures a temporal path among vertices from different variable gadgets. Observe that the starting vertex s_i of X_i is at distance 10 from the ending vertices $t_j^{\ell'}$ ($\ell' \in \{1, 2, 3\}$) of X_j. Therefore, there must be a temporal path using all labels from 1 to 10, to connect them. This path must use the inter-variable edge of the form $e_i\overline{f_j}^{\ell'}$, as any other path is longer than 10. Since the path must be traversed in both directions, each edge $e_i\overline{f_j}^{\ell'}$ ($\ell' \in \{1, 2, 3\}$) must have at least 2 labels. The same holds for the (s_j, t_i^ℓ)-paths ($\ell \in \{1, 2, 3\}$) and the edges $\overline{e_j} f_i^\ell$ ($\ell \in \{1, 2, 3\}$).

These temporal paths ensure that all of the fork vertices in $F^\ell X_j$ are reachable by the vertices of the base BX_i, and vice versa.

The existence of a temporal path from s_i to s_j requires a label on the edge $d_i\overline{d_j}$, as any other (s_i, s_j)-path is longer than 10. Thus, at least one additional label must be assigned to the edge $d_i\overline{d_j}$. Furthermore, since s_i is at distance 4 to d_i and $\overline{d_j}$ is at distance 4 to s_j, label on the edge $d_i\overline{d_j}$ must be 5 or 6. Since these temporal paths do not pass through any of the e vertices of BX_i and BX_j, we still need to ensure that there is a temporal path from e_i to $\overline{e_j}$ and s_j, and vice versa (because of the symmetry this is enough to argue that all of the base vertices of BX_i and BX_j reach each other). Note that e_i is at distance 5 from s_i and any of the t_i^ℓ in X_i, and it is on a shortest (s_i, t_i^ℓ) and (t_i^ℓ, s_i)-temporal path. The same holds for the shortest (temporal) path between s_i and t_j^ℓ. Therefore, the edges $e_i d_i, e_i f_i^\ell$ and $e_i\overline{f_j}^\ell$ must have labels 5 and 6. Similarly, the edges $\overline{e_j}\overline{d_j}, \overline{e_i}\overline{f_j}^\ell$ and $\overline{e_i} f_i^\ell$ must have labels 5 and 6. From the above, we get that there is a temporal path from e_i to $\overline{e_j}$, namely a path through vertices $e_i, \overline{f_j}^\ell, \overline{e_j}$ that uses the labels 5 and 6. Similarly there is a temporal path from $\overline{e_j}$ to e_j using vertices $\overline{e_j}, \overline{f_j}^\ell, e_i$ with labels 5 and 6. What remains to show is that there must be at least one extra label, to ensure the existence of (e_i, s_j) and (s_j, e_i)-temporal paths.

Observe that there are two potential types of paths from e_i to s_j, each of length at most 10. The first type of path that uses one of the f vertices (either f_i^ℓ in X_i or $\overline{f_j}^\ell$ in X_j), and then continues through $\overline{e_j}, \overline{d_j}, \ldots, s_j$. The second type of path uses the edge $d_i\overline{d_j}$. For the first possibility, suppose w.l.o.g. that the temporal path from e_i to s_j travels through vertices $e_i, f_i^1, \overline{e_j}, \overline{d_j}, \ldots, s_j$. We know that the part of this temporal path that travels from $\overline{e_j}$ to s_j, must be labeled with $6, 7, 8, 9, 10$. Since edges $e_i f_i^1$ and $f_i^1\overline{e_j}$ have labels 5 and 6, we see that there must be at least 1 more label, so that e_i reaches s_j via this path (indeed if the edge $e_i f_i^1$ has a label 4 or less, then there is a temporal path from e_i to s_j). Now, for a temporal path from s_j to e_i, of the same form, there must be at least one extra label (indeed, if the edge $f_i^1 e_i$ admits a label 7 or more, there is a temporal path from s_j to e_i). For the second possibility of a path from e_i to s_j (i. e., a path from e_i to s_j that uses the edge $d_i\overline{d_j}$), we get that the (e_i, s_j)-temporal path travels through vertices

$e_i, d_i, \overline{d_j}, \overline{c_j}, \ldots, s_j$. Note that the edge $e_i d_i$ must have labels 5 and 6 and that the part of the path from $\overline{d_j}$ to s_j must have labels $7, 8, 9, 10$, respectively. Now if the edge $d_i\overline{d_j}$ is labeled with 6, there is a temporal path from e_i to s_j. Using the same argument for the (s_j, e_i)-temporal path, we conclude that the edge $d_i\overline{d_j}$ must be labeled also with 5. That is, $d_i\overline{d_j}$ must be labeled with both 5 and 6. To sum up, to ensure the existence of a temporal path among two vertices from two variable gadgets that do not share a fork, a labeling must use at least $2 \cdot (3 + 3) + 2 = 14$ extra labels on the inter-variable edges.

Lastly, let X_j be a variable gadget in G_ϕ that shares a fork with X_i. W.l.o.g. we can suppose that $F^1X_i = F^1X_j$. By the construction of G_ϕ, there exists a temporal path between all vertices in the fork $F^1X_i = F^1X_j$ and all vertices in X_i and X_j. As observed, these paths do not use the inter-variable edges. Using the same arguments as in the case when X_i and X_j do not share a fork, we get that a minimum labeling must use at least $2 \cdot (2 + 2) + 2 = 10$ labels on the inter-variable edges.

The only thing left to study now is what happens in the intersecting fork. We distinguish the following two cases.

- The variable gadget X_j is right-aligned. Then, by the construction of G_ϕ, the fork $F^1X_i = F^1X_j$ is labeled using the same labeling as in the variable gadget X_i. This "saves" 16 labels from the total number of labels used on variable gadgets X_i and X_j.

- The variable gadget X_j is left-aligned. In this case, each edge in the fork $F^1X_i = F^1X_j$ admits two labels. This "saves" only 8 labels from the total number of labels used on variable gadgets X_i and X_j.

From the labeling λ_ϕ of G_ϕ, we construct a truth assignment τ of ϕ as follows. If a variable gadget X_i is left-aligned, we set x_i to True and if it is right-aligned, we set x_i to False.

Suppose that the labeling λ_ϕ satisfies exactly k^* clauses. As previously noted, λ_ϕ uses at least 74 labels on each variable gadget. Whenever two variable gadgets X_i, X_j do not appear in the same clause we need at least 14 extra labels on the inter-variable edges, and whenever X_i, X_j appear in the same clause we need at least 10 labels on the inter-variable gadgets. In the case where X_i, X_j appear in the same clause and both X_i and X_j are left-aligned (i.e. clause of ϕ is not satisfied) the common fork results in 8 less labels, while in the case where X_i is left-aligned and X_j is right-aligned (i.e. clause of ϕ is satisfied), the common fork results in 16 less labels. Consequently,

$$\begin{aligned} |\lambda_\phi| &\geq 74n + 14\binom{n}{2} - 14m + 10m - 8(m-k^*) - 16k^* \\ &= 67n + 7n^2 - 12m - 8k^* \\ &= 7n^2 + 49n - 8k^*. \end{aligned}$$

In the above derived equation, we used the fact that ϕ has $m = \frac{3}{2}n$ clauses. Since $|\lambda_\phi| = \text{OPT}_{\text{MAL}}(G_\phi, d_\phi) \leq 7n^2 + 49n - 8k$ by the statement of the lemma, it follows that $k^* \geq k$, i. e., λ_ϕ satisfies at least k clauses of ϕ. $\square$

MAL is clearly in NP since temporal connectivity can be checked in polynomial time [84]. Hence, the next theorem follows directly from Theorem 5.3.1 and Lemmas 5.3.3 and 5.3.5.

Theorem 5.3.6. *MAL is NP-complete on undirected graphs, when the required maximum age is equal to the diameter of the input graph.*

5.4 The Steiner-tree variations of the problem

In this section, we investigate the computational complexity of the Steiner-Tree variations of the problem, namely MSL and MASL. We start by proving that the age-unrestricted problem MSL remains NP-hard, using a reduction from Vertex Cover. Finally, using a parameterized reduction from Multicolored Clique, we prove that the age-restricted version MASL is W[1]-hard with respect to the number k of labels, even if the maximum allowed age is a constant.

5.4.1 Computational hardness of MSL

In this section, we prove that MSL is NP-complete.

Theorem 5.4.1. *MSL is* NP*-complete.*

Proof. MSL is contained in NP, since temporal connectivity can be checked in polynomial time [84]. To prove that the MSL is NP-hard we provide a polynomial-time reduction from the NP-complete VERTEX COVER problem [80].

VERTEX COVER

Input:	A static graph $G = (V, E)$, a positive integer k.
Question:	Does there exist a subset of vertices $S \subseteq V$ such that $\lvert S\rvert = k$ and $\forall e \in E, e \cap S \neq \emptyset$.

Let (G, k) be an input of the VERTEX COVER problem and denote $|V(G)| = n, |E(G)| = m$. We assume w.l.o.g. that G does not admit a vertex cover of size $k-1$ [80]. We construct (G^*, R^*, k^*), the input of MSL using the following procedure (for an illustration see Figure 5.6). The vertex set $V(G^*)$ consists of the following vertices:

- two starting vertices $N = \{n_0, n_1\}$,
- a "vertex-vertex" corresponding to every vertex of G: $U_V = \{u_v \mid v \in V(G)\}$,
- an "edge-vertex" corresponding to every edge of G: $U_E = \{u_e \mid e \in E(G)\}$,
- $2n + 2m(6k + m)$ "dummy" vertices.

The edge set $E(G^*)$ consists of the following edges:

- an edge between starting vertices, i. e., $n_0 n_1$,
- a path of length 3 between a starting vertex n_1 and every vertex-vertex $u_v \in U_V$ using 2 dummy vertices, and
- for every edge $e = vw \in E(G)$ we connect the corresponding edge-vertex u_e with the vertex-vertices u_v and u_w, each with a path of length $6k+m+1$ using $6k + m$ dummy vertices.

We set $R^* = \{n_0\} \cup U_E$ and $k^* = 6k + 2m(6k + m + 1) + 1$. This finishes the construction. Note that G^* is a graph with $3n + m + 2m(6k + m) + 2$ vertices and

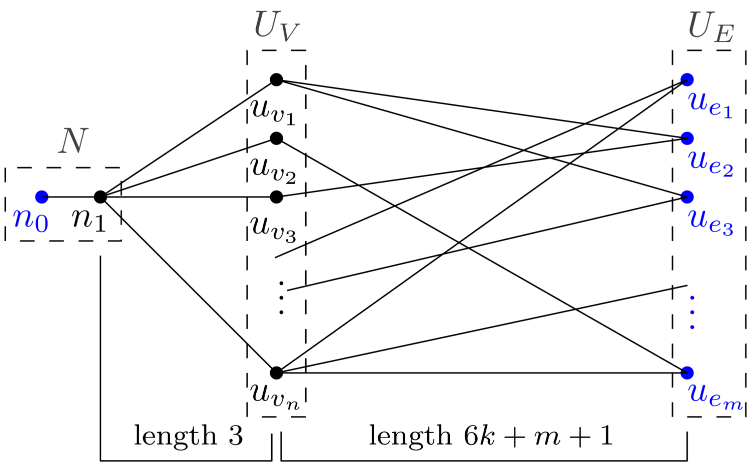


Figure 5.6: Illustration of the MSL instance produced by the reduction presented in the proof of Theorem 5.4.1.

$1 + 3n + 2m(6k + m + 1)$ edges. It is not hard to see that the described construction can be performed in polynomial time.

We claim that (G, k) is a YES instance of the Vertex Cover if and only if (G^*, R^*, k^*) is a YES instance of the MSL.

($\Rightarrow$): Assume (G, k) is a YES instance of the Vertex Cover and let $S \subseteq V(G)$ be a vertex cover for G of size k. We construct a labeling λ for G^* that uses k^* labels and admits a temporal path between all vertices from R^* as follows.

For the sake of easier explanation, we use the following terminology. A temporal path starting at n_0 and finishing at some u_e is called a *returning path*. Contrarily, a temporal path from some u_e to n_0 is called a *forwarding path*.

Let U_S be the set of corresponding vertices to S in G^*. From each edge-vertex u_e there exists a path of length $6k + m + 1$ to at least one vertex $u_v \in U_S$, since S is a vertex cover in G. We label exactly one of these paths, using labels $1, 2, \ldots, 6k + m + 1$. Doing this for all vertices $u_e \in U_E$ we use $m(6k+m+1)$ labels. Now we label a path from each $v \in U_S$ to n_1 using labels $6k + m + 2, 6k + m + 3, 6k + m + 4$. Each path uses 3 labels, and since S is of size k we used $3k$ labels for all of them. At the end, we label the edge n_0n_1 with the label $\ell^* = 6k + m + 5$. Using this procedure we have created a forwarding path from each edge-vertex u_e to the start vertex n_0 and we used $3k + m(6k + m + 1) + 1$ labels.

To create the returning paths, we label paths from n_1 to each vertex in U_S with labels $\ell^*+1, \ell^*+2, \ell^*+3$. Now again, we label exactly one path from vertices in U_S to each edge-vertex u_e, using labels $\ell^*+4, \ell^*+5, \ldots, \ell^*+4+6k+m$. We used extra $3k+m(6k+m+1)$ labels and created a returning path from n_0 to each vertex in U_E.

Altogether, the constructed labeling uses $k^* = 6k+2m(6k+m+1)+1$ labels. What remains to show is that there exists a temporal path between any pair of edge-vertices $u_e, u_f \in U_E$. We can construct a temporal walk W (possibly visiting the same vertex multiple times) from u_e to u_f as follows. Starting at u_e, we go along the forwarding path from u_e to n_0 until we reach n_1. By construction, we arrive at n_1 at time ℓ^*-1. Now consider the returning path from n_0 to u_f. This path goes through n_1 and, by construction, arrives at n_1 at time ℓ^*. Hence, we can extend the temporal walk W from n_1 to u_f by following the returning path from u_1 onward.

($\Leftarrow$): Assume that (G^*, R^*, k^*) is a YES instance of the MSL. We construct a vertex cover of size at most k for G as follows.

Consider the temporal paths connecting n_0 to the vertices in U_E. By the construction of G^* each temporal path from n_0 to a vertex in U_E passes through the set U_V. Hence, for each vertex $u_e \in U_E$ there is some vertex $u_v \in U_V$ such that u_v is temporally connected to u_e. Now consider the temporal paths connecting the vertices in U_E to n_0. Similarly to the argument above, by the construction of G^* each temporal path from a vertex in U_E to n_0 passes through the set U_V. Hence, each vertex $u_e \in U_E$ needs to be temporally connected to some vertex in $u_v \in U_V$. Fix some $u_e \in U_E$. We can conclude that there is a $u_v \in U_V$ such that u_e is temporally connected to u_v by a temporal path of length $6k+m+1$. Furthermore, there is an $u_{v'} \in U_V$ such that u_v' is temporally connected to u_e by a temporal path of length $6k+m+1$. If $u_v \neq u_{v'}$, then we attribute $12k+m+2$ labels to vertex u_e. However, if $u_v = u_{v'}$, then the temporal path of length $6k+m+1$ from u_e to u_v and the temporal path of length $6k+m+1$ from u_v to u_e may share one time edge: Let P_{ve} be the unique path in G^* of length $6k+m+1$ that connects u_v and u_e. Then the (u_v, u_e)-temporal path (resp. (u_e, u_v)-temporal path) traverses the edges of P_{ve} from u_v (resp. u_e) to u_e (resp. u_v), where the edges of P_{ve} are labeled strictly increasingly. Hence the two temporal paths may share at most one time edge. Therefore, in this case, we attribute at least $12k + 2m + 1$ labels to u_e. Overall, we attribute at least $m(12k + 2m + 1)$ labels to the vertices in U_E.

For a vertex $u_e \in U_E$, we call a temporal path from u_e to some $u_v \in U_V$ of length $6k + m + 1$ a forwarding path F_e for u_e. Similarly, we call a temporal path from some $u_{v'}$ to u_e of length $6k + m + 1$ a returning path R_e for u_e. For every u_e we have exactly one forwarding path and one returning path. This is true since every additional path would require at least an additional $6k + m$ labels on the edges between U_V and U_E, and then at most 1 label could be placed on the remaining edges, which would result in no temporal paths between $\{n_0, n_1\}$ and U_V.

This allows us to make the following observation. We define a partial order $<_{\text{label}}$ on the set $\mathcal{P} = \{F_e, R_e \mid e \in E\}$ of forwarding and returning paths as follows. For two paths $P, Q \in \mathcal{P}$, we say that $P <_{\text{label}} Q$ if all labels used in P are strictly smaller than the smallest label used in Q. We can observe that for any two $e, e' \in E$ with $e \neq e'$ we have that $F_e <_{\text{label}} R_{e'}$ since in order for u_e to reach $u_{e'}$, the path F_e needs to be used before the path $R_{e'}$. It follows that there is at most one edge $e \in E$ such that $R_e <_{\text{label}} F_e$ or R_e and F_e are incomparable with respect to $<_{\text{label}}$, otherwise we would reach a contradiction to the above observation. From this we we can deduce that we attribute $12k + 2m + 2$ labels to each of the edges $e \in E$ with $F_e <_{\text{label}} R_e$, since F_e and R_e cannot share any label. Furthermore, there is at most one edge to which we can attribute $12k + 2m + 1$ labels, since, as argued earlier, if F_e and R_e are incomparable with respect to $<_{\text{label}}$, they can share at most one time edge. It follows now that we attribute at least $m(12k + 2m + 2) - 1$ labels to the vertices in U_E.

We now conclude that there are at most $6k+2$ labels used on the edges connecting the sets N and U_V. Next, we identify two (potentially intersecting) subsets of U_V. We specify $U_V^+ \subseteq U_V$ such that $u_v \in U_V^+$ if and only if there exists a returning path R_e for some $e \in E$ that starts in u_v. Similarly, we specify $U_V^- \subseteq U_V$ such that $u_v \in U_V^-$ if and only if there exists a forwarding path F_e for some $e \in E$ that ends in u_v. Assume that $|U_V^+| \leq |U_V^-|$ (the case where $|U_V^+| > |U_V^-|$ is symmetric). We claim that $S = \{v \mid u_v \in U_V^+\}$ is a vertex cover of size at most k for G. It is straightforward to see that S is a vertex cover for G: By the definition of U_V^+, for every $e \in E$ there is a returning path R_e starting in a vertex u_v such that v is one of the two endpoints of e. Hence, for every edge $e \in E$, one of its endpoints is contained in S. In the remainder, we show that $|S| \leq k$.

To this end, consider the temporal connections from n_1 to the vertices in U_E. Every edge-vertex $u_e \in U_E$ is temporally reachable from exactly one vertex-vertex $u_v \in U_V$ through the returning path R_e. Hence, there needs to be a temporal path from n_1 to u_v that arrives in u_v sufficiently early, such that it can be extended to u_e via the returning path R_e. We call this path from n_1 to u_v the *short returning path* R'_e of e. Similarly, consider the temporal connections from the vertices in U_E to n_1. Every edge-vertex $u_e \in U_E$ can reach exactly one vertex $u_v \in U_V$ via a forwarding path F_e. In order to reach n_0, there needs to be a temporal path from u_v to n_1 that starts sufficiently late, such that it can extend the forwarding path from u_e. We call this path from u_v to n_1 the *short forwarding path* F'_e of e.

Analogous to before, we define a partial order $<_{\text{label}}$ on the set $\mathcal{P}' = \{F'_e, R'_e \mid E \in E\} \cup \mathcal{P}$ of (short) forwarding and (short) returning paths. For two paths $P, Q \in \mathcal{P}'$, we say that $P <_{\text{label}} Q$ if all labels used in P are strictly smaller than the smallest label used in Q. Now consider two edges $e, f \in E$ such that the forwarding path F_e ends in u_v and the returning path R_f starts in $u_{v'}$ with $v \neq v'$. Then, by the construction of G^*, there must be a temporal path P from u_v to n_1 and a temporal path P' from n_1 to $u_{v'}$, such that F_e, P, P', and R_e can be concatenated to a temporal path from u_e to u_f. We can assume w.l.o.g. that $P = F'_e$ and $P' = R'_f$. It follows that we must have $F_e <_{\text{label}} F'_e <_{\text{label}} R'_f <_{\text{label}} R_f$ whenever the end vertex u_v of F_e is different from the start vertex $u_{v'}$ of R_f. Next, we categorize the short forwarding and short returning paths by their start and end vertices, respectively. Define $\mathcal{F}'_v = \{F'_e \mid F'_e \text{ starts at } u_v\}$ and $\mathcal{R}'_v = \{R'_e \mid R'_e \text{ ends at } u_v\}$. From what we proved above it follows that for any $P \in \mathcal{F}'_v$ and $Q \in \mathcal{R}'_{v'}$, where $v \neq v'$, we must have $P <_{\text{label}} Q$.

Assume now for contradiction that $|S| > k$ which means that $|U_V^+| > k$ and $|U_V^-| > k$. We analyze the case where for all $u_v \in U_V^-$ we have $|\mathcal{F}'_v| = 1$ and for all $u_v \in U_V^+$ we have $|\mathcal{R}'_v| = 1$ and show that already this case yields a contradiction. From here on we denote $\mathcal{F}'_v = \{F'_v\}$ and $\mathcal{R}'_v = \{R'_v\}$. Similarly, as in arguments we

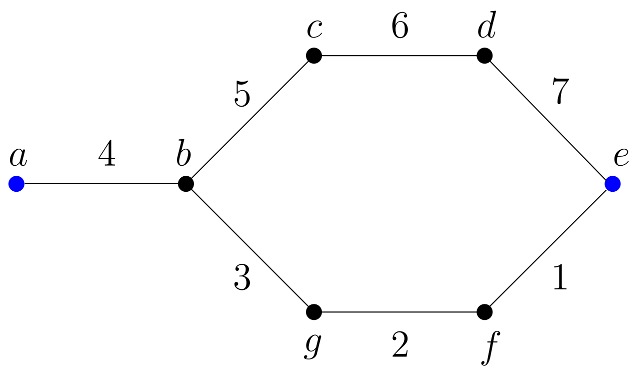


Figure 5.7: An example of an optimal labeling of an MSL instance, where the temporal (sub)-graph connecting terminal vertices $R = \{a, e\}$ is neither a tree nor a tree with a C_4. Note that this is not a solution to ML, as for example there is no temporal path from c to a or to g. Now, we can remove the labels from the edges bg, gf, fe and add them (in the same order) to the edges bc, cd, de, respectively. This way, we obtain an optimal solution, where the subgraph which has labeled edges is a tree (in this case even a path).

made before, we have that for at most one $v \in V$ we can have that $R'_v <_{\text{label}} F'_v$ or R'_v and F'_v are incomparable with respect to $<_{\text{label}}$. Hence, at most one pair of paths, R'_v and F'_v can share a time edge. Since $|U_V^+| > k$ and $|U_V^-| > k$ implies that there are at least $2k+2$ paths, we have that $2k$ paths need three labels each and at most one pair of paths needs five labels in total. However, this yields a number of at least $6k+5$ labels, which is more than the $6k+2$ labels available for these paths. Hence, the assumption that $|S| > k$ leads to a contradiction, which proves that S really is a vertex cover of size at most k. $\square$

5.4.2 Parameterized hardness of MSL and MASL

In this thesis we skip the detailed proof of the following result, and invite the interested reader to check the full paper [88].

Theorem 5.4.2. *MSL is in FPT when parameterized by the number $|R|$ of terminals.*

The main idea of the proof is as follows. From Theorem 5.2.5 and Bumby [23] we know that an optimal solution for ML is a spanning tree, and potentially one further edge that forms a C_4 with the edges of the spanning tree. Note however that, in the case of MSL, we have a weaker requirement on labelings, namely that *only terminal* vertices need to be temporally connected, instead of *all vertices* as in the case of ML. Therefore, in MSL we have an additional difficulty: can the abundance of non-terminal vertices (i. e., of vertices that do not need to be temporally connected) lead to a solution for an MSL instance that is neither a tree nor a tree with a C_4, but still has fewer labels than any solution that is either a tree or a tree with a C_4? As we prove in [88], this cannot happen, i. e., also in the case of MSL it suffices to search for solutions that have this special topological structure. To do so, we specify how an arbitrary optimal solution for MSL (see the example of Figure 5.7 for an illustration) can be transformed into another optimal solution that is a tree or a tree with a C_4. This insight allows us to use an FPT-algorithm for Steiner Tree parameterized by the number of terminals [40] to reveal a subgraph of the MSL instance that we can optimally label using Theorem 5.2.5. Since the number of terminals in the created Steiner Tree instance is larger than the number of terminals in the MSL instance by at most a constant, we obtain an FPT-algorithm for MSL parameterized by the number of terminals.

Note that, since MASL generalizes both MSL and MAL, NP-hardness of MASL is already implied by both Theorems 5.3.6 and 5.4.1. We now prove that MASL is W[1]-hard when parameterized by the number k of labels, even if the restriction a on the age is a constant. Note that the number of terminals can be upper-bounded by a function of the number of labels, since by Theorem 5.2.5 we know that to temporally connect $|R|$ at least $2|R|-4$ labels are necessary. Hence, our results also imply that MASL is W[1]-hard when parameterized by the number $|R|$ of terminals, even if the restriction a on the age is a constant.

To show our parameterized hardness result, we provide a parameterized reduction from Multicolored Clique. This, together with Theorem 5.4.2, implies that MASL is strictly harder than MSL (parameterized by the number $|R|$ of terminals), unless FPT=W[1].

Theorem 5.4.3. *MASL is W[1]-hard when parameterized by the number k of labels, even if the restriction a on the age is a constant.*

Proof. To prove that the MASL is W[1]-hard when parameterized by the number of labels, even if the restriction on the age is a constant, we provide a parameterized polynomial-time reduction from Multicolored Clique parameterized by the number of colors, which is W[1]-hard [48].

Multicolored Clique

Input: A static graph $G = (V, E)$, a positive integer k, a vertex-coloring $c : V(G) \to \{1, 2, \dots, k\}$.

Question: Does G have a clique of size k including vertices of all k colors?

Let (G, k, c) be an input of the Multicolored Clique problem and denote $|V(G)| = n, |E(G)| = m$. We construct (G^*, R^*, a^*, k^*), the input of MASL using the following procedure (for an illustration see Figure 5.8). The vertex set $V(G^*)$ consists of the following vertices:

- a "color-vertex" corresponding to every color of $V(G)$: $C = \{c_i \mid i \in \{1, 2, \dots, k\}\}$,
- a "vertex-vertex" corresponding to every vertex of G: $U_V = \{u_v \mid v \in V(G)\}$,
- an "edge-vertex" corresponding to every edge of G: $U_E = \{u_e \mid e \in E(G)\}$,
- a "color-combination-vertex" corresponding to a pair of two colors of $V(G)$: $W = \{c_{i,j} \mid i, j \in \{1, 2, \dots, k\}, i < j\}$, and
- $2n + 4m + 5m + \frac{11}{8}(k^4 - 2k^3 - k^2 + 2k) + \frac{11}{2}(k^3 - 3k^2 + 2k)$ "dummy" vertices.

The edge set $E(G^*)$ consists of the following edges:

- a path of length 3 (using 2 dummy vertices) between a color-vertex c_i, corresponding to the color i, and every vertex-vertex $u_v \in U_V$, where v is of color i in $V(G)$, i. e., $c(v) = i$,
- for every edge $e = vw \in E(G)$, where $c(v) = i$ and $c(w) = j$, we connect the corresponding edge-vertex u_e with
  - the vertex-vertices u_v and u_w, each with a path of length 3 (using 2 dummy vertices),
  - the color-combination-vertex $c_{i,j}$, with a path of length 6 (using 5 dummy vertices),

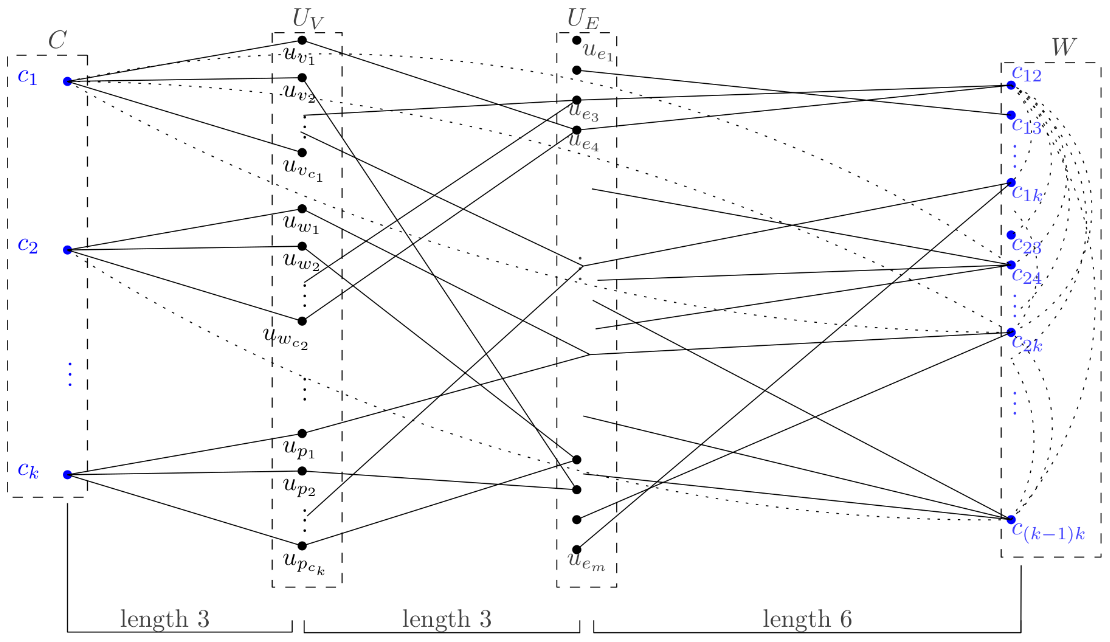


Figure 5.8: Illustration of the MASL instance produced by the reduction presented in the proof of Theorem 5.4.3. For better readability, some paths among the vertices in W and paths among $c_i \in C$ and $c_{j,k} \in W$ $(i \neq j \neq k)$, are not depicted.

- a path of length 12 (using 11 dummy vertices), between each pair of color-combination-vertices, and
- a path of length 12 (using 11 dummy vertices), between all pairs of color-vertices c_i and color-combination-vertices $c_{j,k}$, where $i \notin \{j, k\}$, i. e., we connect the color-vertex of color i with all color-combination vertices of pairs of color that do not include i.

We set $R^* = C \cup W$, $a^* = 12$ and $k^* = 6k + 6(k^2 - k) + 6(k^2 - k) + 3(k^4 - 2k^3 - k^2 + 2k) + 12(k^3 - 3k^2 + 2k)$. Note that $k^* \in O(k^4)$, hence the parameter number of labels of the MASL instance is upper-bounded by a function of k. Furthermore, observe that the restriction on the age is a constant. This finishes the construction. It is not hard to see that this construction can be performed in polynomial time. At the end G^* is a graph with $3n + 10m + \frac{1}{2}(k^2 + k) + \frac{11}{8}(k^4 - 2k^3 - k^2 + 2k) + \frac{11}{2}(k^3 - 3k^2 + 2k)$ vertices and $3n + 12m + \frac{3}{2}(k^4 - 2k^3 - k^2 + 2k) + 6(k^3 - 3k^2 + 2k)$ edges.

We claim that (G, k, c) is a YES instance of the Multicolored Clique if and only if (G^*, R^*, a^*, k^*) is a YES instance of the MASL.

($\Rightarrow$): Assume (G, k, c) is a YES instance of the Multicolored Clique. Let

$S \subseteq V(G)$ be the set of vertices that form a multicolored clique in G. We construct a labeling λ for G^* that uses k^* labels, which are not larger than $a^* = 12$, and admits a temporal path between all vertices from R^* as follows.

Let U_S be the set of corresponding vertices to S in G^*. For each $v \in S$ of color i we label the three edges connecting c_i to u_v with labels $1, 2, 3$, one per each edge, in order to create temporal paths starting in c_i and with labels $12, 11, 10$, one per each edge, in order to create temporal paths that finish in c_i. For every edge $vw = e \in E$ with endpoints in S we label the path from both of its endpoint vertex-vertices u_v, u_w to the edge-vertex u_e with labels $4, 5, 6$, one per each edge, and with labels $9, 8, 7$, one per each edge. This ensures the existence of both temporal paths between c_i and c_j. More precisely, (c_i, c_j)-temporal path (resp. (c_j, c_i)-temporal path) uses labels $1, 2, 3$ to reach u_v (resp. u_w), from where it continues with $4, 5, 6$ to u_e, then with $7, 8, 9$ reaches u_w (resp. u_v) and finally with $10, 11, 12$ it finishes in c_j (resp. c_i). Note that since S is a multicolored clique then each vertex $v' \in S$ is of a unique color i' and all vertices in S are connected. Therefore, using the above construction for all vertices in S, vertex c_i reaches and is reached by every other color vertex c_j through the vertex-vertex u_v. Even more, since there is an edge e connecting any two vertices $v, w \in S$, there is a unique edge-vertex u_e (and consequently a unique path), that is used for both temporal paths between vertex-vertices u_v, u_w and their corresponding color-vertices. The above construction clearly produces a temporal path (of length 12) between any two color-vertices. This construction uses $2 \cdot 3$ labels between every color-vertex c_i and its unique vertex-vertex u_v, where $v \in S$ and $c(v) = i$, and $2 \cdot 6$ labels from each edge-vertex u_e to both of its endpoint vertex-vertices, where e is an edge of the multicolored clique formed by the vertices in S. All in total we used $6k + 12\binom{k}{2} = 6k + 6(k^2 - k)$ labels, to connect all edge-vertices corresponding to edges formed by S with their endpoints vertex-vertices.

Now, let $c_{i,j}$ and $c_{i',j'}$ be two arbitrary color-combination-vertices. By the construction of G^* there is a unique path of length 12 connecting them, which we label with labels $1, 2, \ldots, 12$ in both directions. This labeling uses $2 \cdot 12$ labels for each pair of color-combination-vertices, hence all together we use $24\frac{|W|(|W|-1)}{2}$ labels, since $|W| = \binom{k}{2}$ this is equal to $3(k^4 - 2k^3 - k^2 + 2k)$.

Finally, let $c_{i'}$ and $c_{i,j}$ be two arbitrary color and color-combination-vertices, respectively. In the case when $i' \notin \{i,j\}$ there is a unique path of length 12 in G^* between them (that uses only the dummy vertices). We label this path with labels $1, 2, \ldots, 12$ in both directions. This procedure uses $2 \cdot 12$ labels for each pair of such vertices, hence all together we use $24k\binom{k-1}{2}$ labels, which equals $12(k^3-3k^2+2k)$. In the case when $i' \in \{i,j\}$ (w.l.o.g. $i' = i$) we connect the vertices using the following path. In S there exists a unique vertex of color i, denote it v. By the definition of S there is also vertex $w \in S$ of color j, which is connected to v with some edge, denote it e. Therefore, to obtain a $(c_i, c_{i,j})$-temporal path, we first reach u_v from c_i with labels $1, 2, 3$, then continue to u_e, using labels $4, 5, 6$, from where we continue to $c_{i,j}$ using the labels $7, 8, \ldots, 12$. The $(c_{i,j}, c_i)$-temporal path uses the same edges, with labels in reversed order. This construction introduced $2 \cdot 6$ new labels on the path of length 6 between the edge-vertex u_e and the color-combination-vertex c_{ij} and reused all labels on the (c_i, u_e)-temporal paths. Repeating this for every color-combination-vertex we use $2 \cdot 6|W|$ new labels, since $|W| = \binom{k}{2}$ this is equal to $6(k^2 - k)$.

All together λ uses $6k+6(k^2-k)+6(k^2-k)+3(k^4-2k^3-k^2+2k)+12(k^3-3k^2+2k)$ labels.

($\Leftarrow$): Assume that (G^*, R^*, a^*, k^*) is a YES instance of the MASL and let λ be the corresponding labeling of G^*. Before we construct a multicolored clique for G, we prove that the distance between any two terminal vertices from R^* in G^* is 12.

Case A. Let $c_i, c_j \in C$ be two arbitrary color-vertices and let e be an edge in G with endpoints of color i and j, i. e., $e = vw \in E(G)$ and $c(v) = i, c(w) = j$. There are two options on how to reach c_j from c_i. One when the path connecting them passes through the set U_E and the other when it passes through the set W.

Case A-1. If the path passes through the set E, we must first go through a vertex-vertex u_v, then we go to the edge-vertex u_e, continue to the vertex-vertex u_w and finish in c_j. Since all these vertices are connected with a path of length 3, we get that the distance of the whole (c_i, c_j)-path is 12.

Case A-2. If the path passes through the set W, then we must go through the color-combination-vertex $c_{i,j}$. Since the path between any color-vertex and color-combination-vertex is of length 12 (we prove this in the following paragraph), the whole (c_i, c_j)-path is of length 24.

Therefore, the shortest path connecting two color-vertices is of length 12 and must go through the appropriate edge-vertex.

Case B. Let $c_{i,j}$ and $c_{i'}$ be two arbitrary vertices from the color-combination-vertices and color-vertices. We distinguish two cases.

Case B-1. First, when $i' \notin \{i, j\}$. Then, by the construction of G^*, there exists a direct path of length 12, connecting them. Any other $(c_{i'}, c_{i,j})$-path must either go from $c_{i'}$ to some color-combination-vertex $c_{i',j'}$, which is then connected with a path of length 12 to the $c_{i,j}$, or go to one of the color-vertices and then continue to the $c_{i,j}$. In both cases, the constructed path is strictly longer than 12.

Case B-2. Second, when $i' \in \{i, j\}$. Let $c(v) = i$ and $vw = e \in E(G)$ be such that $c(w) = j$. Then there is a path from c_i to $c_{i,j}$ that goes through the vertex-vertex u_v (using a path of length 3), continues to the edge-vertex u_e (using a path of length 3), which is connected to the color-combination-vertex $c_{i,j}$ (using a path of length 6). Hence the constructed $(c_i, c_{i,j})$-path is of length 12. There exists also another $(c_i, c_{i,j})$-path, that goes through some other $c_{i,j'}$ color-combination-vertex, but it is longer than 12.

Case C. Let $c_{i,j}$ and $c_{i',j'}$ be two arbitrary color-combination-vertices. By construction of G^*, there is a path of length 12 connecting them. Any other $(c_{i,j}, c_{i',j'})$-path, must use at least one vertex-vertex, which is at a distance 9 from the color-combination-vertices (therefore the path through it would be of length at least 18), or a color-vertex, which is at a distance 12 from the color-combination-vertices. In both cases, the constructed path is strictly longer than 12.

It follows that the distance between any two terminal vertices in R^* is 12, hence a temporal path connecting them must use all labels from 1 to 12. Using this property we know that any labeling that admits a temporal path among each pair of terminal vertices must use all labels $1, 2, \ldots, 12$ on the temporal paths between any two color-combination-vertices $c_{i,j}$ and $c_{i',j'}$, and between a color-vertex $c_{i'}$ and a color-combination-vertex $c_{i,j}$, where $i' \notin \{i, j\}$. This is true as by construction there are unique paths of length 12 connecting each pair of them. For these temporal paths we must use $2 \cdot 12\frac{|W|(|W|-1)}{2}$ labels (since $|W| = \binom{k}{2}$ this is equal to $3(k^4 - 2k^3 - k^2 + 2k)$) and $2 \cdot 12k\binom{k-1}{2}$ labels (which equals $12(k^3 - 3k^2 + 2k)$). Therefore, the labeling λ can use only $6k + 6(k^2 - k) + 6(k^2 - k)$ labels to connect all other terminals.

Let us now observe what happens with the temporal paths connecting the remaining temporal vertices. To create a temporal path starting in a color-vertex c_i and ending in some other color-vertex (or color-combination-vertex), λ must label at least 3 edges to allow c_i to reach one of its corresponding vertex-vertices u_v. Similarly, it holds for a temporal path ending in c_i. Since the path connecting c_i to some other terminal is of length 12, the labels used on the temporal paths starting and ending in c_i cannot be the same. In fact, the labels must be $1, 2, 3$ for one direction and $12, 11, 10$ for the other. Therefore, λ uses at least $6k$ labels on edges between vertices of C and U_V. Extending the arguing from above, for c_i to reach some (suitable) edge-vertex u_e (where v is one of the endpoints of e) the path needs to continue from u_v to u_e and must use the labels $4, 5, 6$ (or $9, 8, 7$ in case of the path in the opposite direction). From u_e the path can continue to the corresponding color-combination-vertex $c_{i,j}$ where it must use the labels $7, 8, \ldots, 12$, or to the vertex-vertex $u_{v'}$ corresponding to the other endpoint of edge e (the edge e is between v and v'). This finishes the construction of the temporal path from a color-vertex to the color-combination-vertex and the temporal paths among color-vertices. It remains to connect a color-combination-vertex with its corresponding color-vertices. The temporal path must go through some edge-vertex u_e, that is at distance 6 from it, therefore the labeling must use the labels $1, 2, \ldots, 6$. From u_e the path continues to the suitable vertex-vertex and then to the color-vertex. Using the above labeling we see that λ must use at least $2 \cdot 6|W|$ labels (which equals $6(k^2 - k)$ labels) on the edges between the color-combination-vertices in W and the edge-vertices in U_E and at least $2 \cdot 6\binom{k}{2}$ labels (which equals $6(k^2 - k)$ labels) on the edges between the edge-vertices in U_E and vertex-vertices in U_V. Since all this together equals k^*, all of the bounds are tight, i. e., labeling cannot use more labels.

We still need to show that for every color-vertex c_i there exists a unique vertex-vertex u_v connected to it such that all temporal paths to and from c_i travel only through u_v. By the argument on the number of labels used, we know that there can be at most two vertex-vertices that lie on temporal paths to or from c_i. More precisely, one that lies on every temporal path starting in c_i and the other (possibly the same) that lies on every temporal path that finishes in c_i. Let now $u_v, u_{v'}$ be two such vertex-vertices. Suppose that u_v lies on all temporal paths that start in c_i and $u_{v'}$ on all temporal paths that end in c_i. Now let u_e be the edge-vertex on a temporal path from c_i to c_j, and let u_w be the vertex-vertex connected to c_j and u_e. Therefore the (c_i, c_j)-temporal path has the following form: it starts in c_i, uses the labels $1, 2, 3$ to reach u_v, then continues to u_e with $4, 5, 6$, then with $7, 8, 9$ reaches u_w and with $10, 11, 12$ ends in c_j. To obtain the $(c_i, c_{i,j})$-temporal path we must label the edges from u_e to $c_{i,j}$ with the labels $6, 7, \ldots, 12$, since the edge-vertex u_e is the only edge-vertex connected to the color-combination-vertex $c_{i,j}$ that can be reached from c_i (if there would be another such edge-vertex, then the labeling λ would use too many labels on the edges between U_V and U_E). Now, for the color-vertex c_j to be able to reach the color-combination-vertex $c_{i,j}$, it must use the same labels between u_e and $c_{i,j}$ (using the same reasoning as before). Therefore the path from c_j to u_e (through) u_w uses also the labels $1, 2, \ldots, 6$. But then for c_j to reach c_i the temporal path must use the vertex-vertex u_w, even more it must use the edge-vertex u_e and consequently the vertex-vertex u_v, from where it would reach c_i. This implies that we must have that $u_v = u_{v'}$. Therefore, every color-vertex c_i admits a unique vertex-vertex u_v that lies on all (c_i, c_j) and (c_j, c_i)-temporal paths. For the conclusion of the proof, we claim that all vertices v corresponding to these unique vertex-vertices u_v of color-vertices c_i, form a multicolored clique in G. This is true as, by construction, a temporal path between two vertex-vertices u_v, u_w corresponds to the edge $vw = e \in E(G)$. Since every vertex-vertex is connected to exactly one color-vertex, this corresponds to the vertex coloring of $V(G)$. In G^* there is a temporal path among any two color vertices, therefore the vertex-vertices used in these temporal paths can be reached among each other, which means that they really do form a multicolored clique. $\square$

5.5 Concluding remarks

In this chapter, we studied four natural temporal labeling problems. We considered the settings where we have an age restriction on the labeling or not. Furthermore, we investigated settings where the labeling has to temporally connect *every* vertex pair and settings where only a given set of *terminal* vertices have to be pairwise temporally connected. One variant (ML) is polynomial-time solvable, whereas the three other variants (MAL, MSL, and MASL) turn out to be NP-hard. For the latter two, we also give parameterized complexity results with respect to the number of labels and to the number of terminals as parameters. Our work spawns several future research directions.

Recall that a labeling λ satisfying MAL with the age restriction being the diameter of the graph, is of the size $O(n^2)$ (see Observation 5.2.1). In Lemma 5.2.2, we show that on cycles, the labeling uses $\Theta(n^2)$ labels. Therefore, it would be interesting to study, for which graph classes the optimal labeling uses $o(n^2)$ or $O(n)$ labels. We show that MAL is NP-complete when the upper age bound is equal to the diameter d of the input graph G. On the other hand, if the upper age bound is $2r$, where r is the radius of G, MAL can be computed in polynomial time. Indeed, using the results of Section 5.2.1, it easily follows that if G contains (or does not contain) a C_4, then the labeling consists of $2n - 4$ (or $2n - 3$) labels. An interesting question that arises now is: For which values of an upper age bound a, where $d \leq a \leq 2r$, can MAL be solved efficiently? Furthermore, it would be interesting to analyse the parameterized complexity of MAL. A canonical starting point would be to consider the number of time labels as a parameter.

Our results for MSL and MASL also leave some open questions and several natural future research directions. Recall that the number k of labels is a larger parameter than the number of terminals. Hence, the parameterized complexity with respect to those two parameters of MSL is resolved. For MASL it remains open whether we can obtain an XP algorithm for those parameters.

More generally, it would be interesting to investigate structural parameterizations for all NP-hard problem variants of this work. We conjecture that all problem variants are polynomial-time solvable if the input graph G is a tree. Consequently, parameters that measure tree-likeness, such as treewidth, are promising candidates for obtaining FPT results.

CHAPTER 6

Realizing Temporal Graphs From Fastest Travel Times

This chapter is the result of a collaborative work with George B. Mertzios, Hendrik Molter, and Paul G. Spirakis.

The full paper, containing our detailed findings, is available as a preprint on ArXiv [89]. The preliminary results were presented in the Proceedings of the 2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND) 2024 [90].

The major part of the proof for Theorem 6.2.2 was contributed by Hendrik Molter; therefore, I have chosen not to present it in this chapter. The complete proof, including all the details, is available in our full paper.

6.1 Introduction

The (static) *graph realization* problem with respect to a graph property $\mathcal{P}$ is to find a graph that satisfies property $\mathcal{P}$, or to decide that no such graph exists. The motivation for graph realization problems stems both from "verification" and from network design applications in engineering. In *verification* applications, given the outcomes of some experimental measurements (resp. some computations) on a network, the aim is to (re)construct an input network which complies with them. If such a reconstruction is not possible, this proves that the measurements are incorrect or implausible (resp. that the algorithm which made the computations is incorrectly implemented). One example of a graph realization (or reconstruction) problem is the recognition of probe interval graphs, in the context of the physical mapping of DNA, where one wants to reconstruct relative positions of fragments of DNA along the genome from certain pairwise overlap information (for more details see [96,97] and [64, Chapter 4]). In *network design* applications, the goal is to design network topologies having a desired property [10, 66]. Analyzing the computational complexity of the graph realization problems for various natural and fundamental graph properties $\mathcal{P}$ requires a deep understanding of these properties. Among the most studied parameters for graph realization are constraints on the distances between vertices [13, 14, 20, 30, 33, 71], on the vertex degrees [12, 45, 63, 65, 70], on the eccentricities [11, 15, 77, 95], and on connectivity [29, 55–57, 59, 65], among others.

In the simplest version of a (static) graph realization problem with respect to vertex distances, we are given a symmetric $n \times n$ matrix D and we are looking for an n-vertex undirected and unweighted graph G such that $D_{i,j}$ equals the distance between vertices v_i and v_j in G. This problem can be trivially solved in polynomial time in two steps [71]: First, we build the graph $G = (V, E)$ such that $v_i v_j \in E$ if and only if $D_{i,j} = 1$. Second, from this graph G we compute the matrix D_G which captures the shortest distances for all pairs of vertices. If $D_G = D$ then G is the desired graph, otherwise there is no graph having D as its distance matrix. Nontrivial variations of this problem have been extensively studied, such as for weighted graphs [71, 111], as well as for cases where the realizing graph has to belong to a specific graph family [13, 71]. Other variations of the problem include the cases where entries of the input matrix D may contain a range of consecutive permissible values [13,113,121], or even an arbitrary set of acceptable values [14] for the distance between the corresponding two vertices.

In this chapter, we consider *periodic* temporal graphs, i. e., temporal graphs in which the temporal availability of each edge of the underlying graph is periodic. Many natural and technological systems exhibit periodic temporal behaviour. For example, in railway networks, an edge is present at a time step t if and only if a train is scheduled to run on the respective rail segment at time t [8]. Similarly, a satellite, which makes pre-determined periodic movements, can establish a communication link (i. e., a temporal edge) with another satellite whenever they are sufficiently close to each other; the existence of these communication links is also periodic. In a railway (resp. satellite) network, a fastest temporal path from u to v represents the fastest railway connection between two stations (resp. the quickest communication delay between two moving satellites). Furthermore, periodicity appears also in (the otherwise quite complex) social networks which describe the dynamics of people meeting [94, 115], as every person follows mostly a weekly routine [8].

Although periodic temporal graphs have already been studied (see [26, Class 8] and [8, 47, 103, 104]), we make here the first attempt to understand the complexity of a graph realization problem in the context of temporal graphs. Therefore, we focus in this work on the most fundamental case, where all edges have the same period Δ (while in the more general case, each edge e in the underlying graph has a period Δ_e). As it turns out, the periodic temporal graph realization problem with respect to a given $n \times n$ matrix D of the fastest duration times has a very different computational complexity behaviour than the classic graph realization problem with respect to shortest path distances in static graphs.

Formally, let $G = (V, E)$ and $\Delta \in \mathbb{N}$, and let $\lambda : E \to \{1, 2, \ldots, \Delta\}$ be an edge-labeling function that assigns to every edge of G exactly one of the labels from $\{1, \ldots, \Delta\}$. Then we denote by (G, λ, Δ) the Δ-*periodic temporal graph* (G, L), where for every edge $e \in E$ we have $L(e) = \{i\Delta + x : i \geq 0, x \in \lambda(e)\}$. In this case, we call λ a Δ-*periodic labeling* of G; see Figure 6.1 for an illustration. When it is clear from the context, we drop Δ from the notation and we denote the (Δ-periodic) temporal graph by (G, λ). To avoid confusion, we would like to point out that the notation for the labeling function in this chapter is slightly different to the one we used so far. What was in the previous chapters denoted as a temporal graph (G, λ) is equivalent to our notation (G, L). And, whenever we use and talk about the labelling λ in this chapter, we mean the Δ-periodic labeling λ that assigns a single value from $\{1, 2, \ldots, \Delta\}$ to each edge.

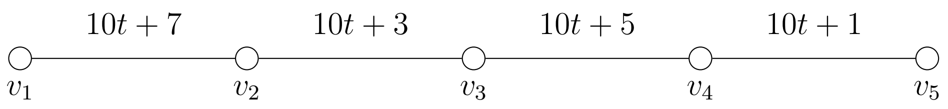


Figure 6.1: An example of a Δ-periodic temporal graph (G, λ, Δ), where $\Delta = 10$ and the 10-periodic labeling $\lambda : E \to \{1, 2, \dots, 10\}$ is as follows: $\lambda(v_1v_2) = 7$, $\lambda(v_2v_3) = 3$, $\lambda(v_3v_4) = 5$, and $\lambda(v_4v_5) = 1$. Here, the fastest temporal path from v_1 to v_2 traverses the first edge v_1v_2 at time 7, the second edge v_2v_3 a time 13, the third edge v_3v_4 at time 15 and the last edge v_4v_5 at time 21. This results in the total duration of $21 - 7 + 1 = 15$ for the fastest temporal path from v_1 to v_5.

Our contribution. We initiate the study of naturally motivated graph realization problems in the temporal setting. Our target is not to model unreliable communication, but instead to *verify* that particular measurements regarding fastest temporal paths in a periodic temporal graph are plausible (i. e., “realizable”). To this end, we introduce and investigate the following problem, capturing the setting described above:

Simple periodic Temporal Graph Realization (Simple TGR)

Input: An integer $n \times n$ matrix D, a positive integer Δ.

Question: Is there a graph $G = (V, E)$ with vertices $\{v_1, \dots, v_n\}$ and a Δ-periodic labeling $\lambda : E \to \{1, 2, \dots, \Delta\}$ such that, for every i, j, the duration of the fastest temporal path from v_i to v_j in the Δ-periodic temporal graph (G, λ, Δ) is $D_{i,j}$?

Given the matrix D, refereed to also as the *duration matrix*, it is easy to observe that, similarly to the static case, if $D_{i,j} = 1$ then v_i and v_j must be connected by an edge. This uniquely defines the graph G, which we call the *underlying graph* of D.

In our work we focus on exact algorithms. We start by showing the NP-hardness of the problem (Theorem 6.2.1), even if Δ is a small constant. To establish a baseline for tractability, we show that Simple TGR is polynomial-time solvable if the underlying graph is a tree (Theorem 6.3.1).

Building upon these initial results, we explore the possibilities to generalize our polynomial-time algorithm using the *distance-from-triviality* parameterization paradigm [49, 67]. That is, we investigate the parameterized computational complexity of Simple TGR with respect to structural parameters of the underlying graph that measure its “tree-likeness”.

We obtain the following results. We show that Simple TGR is W[1]-hard when parameterized by the feedback vertex number of the underlying graph (Theorem 6.2.2). Note that our parameterized hardness result rules out fixed-parameter tractability for several popular graph parameters such as *treewidth*, *degeneracy*, *cliquewidth*, *distance to chordal graphs*, and *distance to outerplanar graphs*. We complement this hardness result by showing that Simple TGR is fixed-parameter tractable (FPT) with respect to the *feedback edge number* k of the underlying graph (Theorem 6.3.2). This result also implies an FPT algorithm for any larger parameter, such as the *maximum leaf number*. A similar phenomenon of getting W[1]-hardness with respect to the feedback vertex number, while getting an FPT algorithm with respect to the feedback edge number, has been observed only in a few other temporal graph problems related to the connectivity between two vertices [27, 44, 58].

Our FPT algorithm works as follows on a high level. First, we distinguish $O(k^2)$ vertices which we call "important vertices". Then, we guess the fastest temporal paths for each pair of these important vertices; as we prove, the number of choices we have for all these guesses is upper bounded by a function of k. Then we also need to make several further guesses (again using a bounded number of choices), which altogether leads us to specify a small (i. e., bounded by a function of k) number of different configurations for the fastest paths between *all pairs* of vertices. For each of these configurations, we must then make sure that the labels of our solution will not allow any other temporal path from a vertex v_i to a vertex v_j have a *strictly smaller* duration than $D_{i,j}$. This naturally leads us to build one Integer Linear Program (ILP) for each of these configurations. We manage to formulate all these ILPs by having a number of variables that is upper-bounded by a function of k. Finally, we use Lenstra's Theorem [92] to solve each of these ILPs in FPT time. In the end, our initial instance is a Yes-instance if and only if at least one of these ILPs is feasible.

The above results provide a fairly complete picture of the parameterized computational complexity of Simple TGR with respect to structural parameters of the underlying graph which measure "tree-likeness". To obtain our results, we prove several properties of the fastest temporal paths, which may be of independent interest.

Related work. There are some problem settings that share similarities with ours, which we discuss now in more detail.

Several problems have been studied where the goal is to assign labels to (sets of) edges of a given static graph in order to achieve certain connectivity-related properties [4,43,87,98]. The main difference to our problem setting is that in the mentioned works, the input is a graph and the sought labeling is not periodic. Furthermore, the investigated properties are temporal connectivity between all vertices [4,87,98], temporal connectivity among a subset of vertices [87], or reducing reachability among the vertices [43]. In all these cases, the duration of the temporal paths has not been directly considered.

Preliminaries and notation. Let $P = (u = v_1, v_2, \ldots, v_p = v)$ be a path from u to v in G. Recall that, in this chapter, every edge has exactly one time label in every period of Δ consecutive time steps. Therefore, as we are only interested in the fastest duration of temporal paths, many times we refer to (P, λ, Δ) as any of the temporal paths from $u = v_1$ to $v = v_p$ along the edges of P, which starts at the edge v_1v_2 at time $\lambda(v_1v_2) + c\Delta$, for some $c \in \mathbb{N}$, and then sequentially visits the rest of the edges of P as early as possible. We denote by $d(P, \lambda, \Delta)$, or simply by $d(P, \lambda)$ when Δ is clear from the context, the duration of any of the temporal paths (P, λ, Δ); note that they all have the same duration. For a pair of vertices $u, v \in V(G)$ we denote by $d(u, v)$ the duration of the fastest temporal path from u to v in (G, λ). Whenever we use the term *label of an edge* e, we actually mean $\lambda(e) \in [\Delta]$. Note that for a given path (P, λ, Δ) that passes through the edge e, the label used by P at that edge is $\lambda(e) + c\Delta$, for some $c \geq 0$. Many times we also refer to a path $P = (u = v_1, v_2, \ldots, v_p = v)$ from u to v in G, as a temporal path in (G, λ, Δ), where we actually mean that (P, λ, Δ) is a temporal path with P as its underlying (static) path.

We remark that a fastest path between two vertices in a temporal graph can be computed in polynomial time [22, 130]. Hence, given a Δ-periodic temporal graph (G, λ, Δ), we can compute in polynomial time the matrix D which consists of durations of the fastest temporal paths among all pairs of vertices in (G, λ, Δ).

6.2 Hardness results for Simple TGR

In this section, we prove that in general, it is NP-hard to determine a Δ-periodic temporal graph (G, λ) respecting a duration matrix D, even if Δ is a small constant.

Theorem 6.2.1. *Simple TGR is* NP*-hard for all* $\Delta \geq 3$*.*

Proof. We present a polynomial-time reduction from the NP-hard problem NAE 3-SAT [116]. We are given a formula ϕ that is a conjunction of so-called NAE (not-all-equal) clauses, where each clause contains exactly 3 literals (with three distinct variables). A NAE clause evaluates to TRUE if and only if not all of its literals are equal, that is, at least one literal evaluates to TRUE and at least one literal evaluates to FALSE. We are asked whether ϕ admits a satisfying assignment.

Given an instance ϕ of NAE 3-SAT, we construct an instance (D, Δ) of Simple TGR as follows.

We start by describing the vertex set of the underlying graph G of D.

- For each variable x_i in ϕ, we create three variable vertices x_i, x_i^T, x_i^F.
- For each clause c in ϕ, we create one clause vertex c.
- We add one additional super vertex v.

Next, we describe the edge set of G.

- For each variable x_i in ϕ we add the following five edges: $\{x_i, x_i^T\}$, $\{x_i, x_i^F\}$, $\{x_i^T, x_i^F\}$, $\{x_i^T, v\}$, and $\{x_i^F, v\}$.
- For each pair of variables x_i, x_j in ϕ with $i \neq j$ we add the following four edges: $\{x_i^T, x_j^T\}$, $\{x_i^T, x_j^F\}$, $\{x_i^F, x_j^T\}$, and $\{x_i^F, x_j^F\}$.
- For each clause c in ϕ we add one edge for each literal. Let x_i appear in c. If x_i appears non-negated in c we add edge $\{c, x_i^T\}$. If x_i appears negated in c we add edge $\{c, x_i^F\}$.

This finishes the construction of G. For an illustration see Figure 6.2.

We set Δ to some constant larger than two, that is, $\Delta \geq 3$. Next, we specify the durations in the matrix D between all vertex pairs. For the sake of simplicity

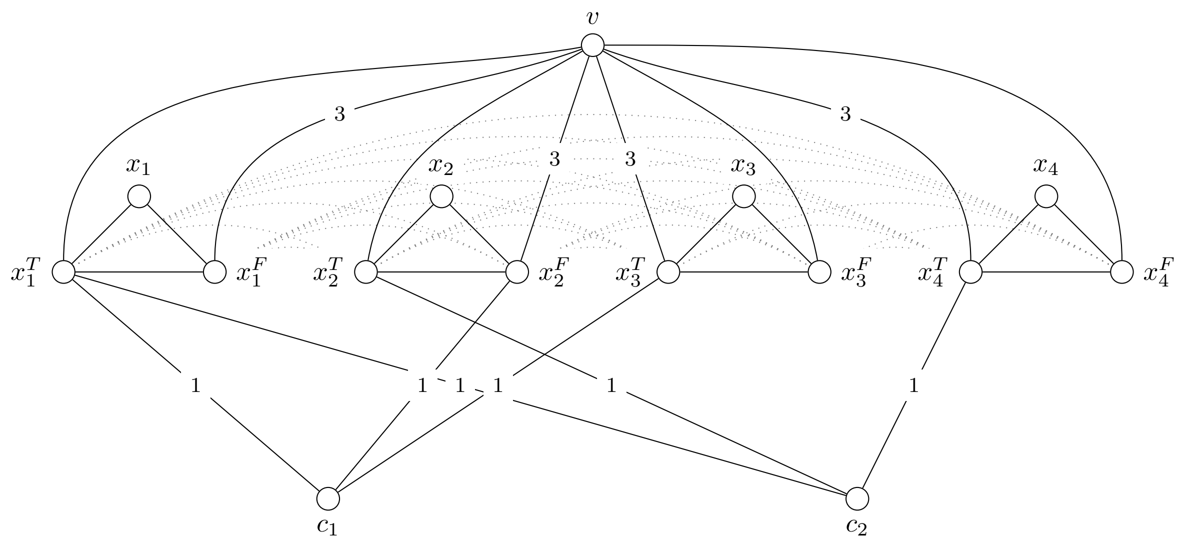


Figure 6.2: Illustration of the temporal graph (G, λ) from the NP-hardness reduction, where the NAE 3-SAT formula ϕ is of the form $\phi = \text{NAE}(x_1, \overline{x}_2, x_3) \wedge \text{NAE}(x_1, x_2, x_4)$. To improve the readability, we draw edges between vertices x_i^T and x_j^F (where $i \neq j$) with grey dotted lines. Presented is the labeling of G corresponding to the assignment $x_1 = x_2 = \text{TRUE}$ and $x_3, x_4 = \text{FALSE}$, where all unlabeled edges get the label 2.

we write $D_{u,v}$ as $d(u, v)$, where u, v are two vertices of G. We start by setting the value of $d(u, v) = 1$ where u and v are two adjacent vertices in G.

- For each variable x_i in ϕ and the super vertex v we specify the following durations: $d(x_i, v) = 2$ and $d(v, x_i) = \Delta$.
- For each clause c in ϕ and the super vertex v we specify the following durations: $d(c, v) = 2$ and $d(v, c) = \Delta - 1$.
- Let x_i be a variable that appears in clause c, then we specify the following durations: $d(c, x_i) = 2$ and $d(x_i, c) = \Delta$. If x_i appears non-negated in c we specify the following durations: $d(c, x_i^F) = 2$ and $d(x_i^F, c) = \Delta$. If x_i appears negated in c we specify the following durations: $d(c, x_i^T) = 2$ and $d(x_i^T, c) = \Delta$.
- Let x_i be a variable that does *not* appear in clause c, then we specify the following durations: $d(x_i, c) = 2\Delta$, $d(c, x_i) = \Delta{+}2$ and $d(c, x_i^T) = d(c, x_i^F) = 2$, $d(x_i^T, c) = d(x_i^F, c) = \Delta$.
- For each pair of variables $x_i \neq x_j$ in ϕ we specify the following durations: $d(x_i, x_j) = 2\Delta + 1$ and $d(x_i, x_j^T) = d(x_i, x_j^F) = \Delta + 1$.

- For each pair of clauses $c_i \neq c_j$ in ϕ we specify the following durations: $d(c_i, c_j) = \Delta + 1$.

This finishes the construction of the instance (D, Δ) of Simple TGR which can clearly be done in polynomial time. In the remainder, we show that (D, Δ) is a Yes-instance of Simple TGR if and only if NAE 3-SAT formula ϕ is satisfiable.

($\Rightarrow$): Assume the constructed instance (D, Δ) of Simple TGR is a Yes-instance. Then there exists a label $\lambda(e)$ for each edge $e \in E(G)$ such that for each vertex pair u, w in the temporal graph (G, λ, Δ) we have that a fastest temporal path from u to w is of duration $d(u, w)$.

We construct a satisfying assignment for ϕ as follows. For each variable x_i, if $\lambda(\{x_i, x_i^T\}) = \lambda(\{x_i^T, v\})$, then we set x_i to true, otherwise we set x_i to false.

To show that this yields a satisfying assignment, we need to prove some properties of the labeling λ. First, observe that adding an integer t to all time labels does not change the duration of any temporal paths. Second, observe that if for two vertices u, w we have that $d(u, w)$ equals the distance between u and w in G (i. e., the duration of the fastest temporal path from u to w equals the distance of the shortest path between u and w), then there is a shortest path P from u to w in G such that the labeling λ assigns consecutive time labels to the edges of P.

Let $\lambda(\{x_i, x_i^T\}) = t$ and $\lambda(\{x_i, x_i^F\}) = t'$, for an arbitrary variable x_i. If both $\lambda(\{x_i^T, v\}) \neq t+1$ and $\lambda(\{x_i^F, v\}) \neq t'+1$, then $d(x_i, v) > 2$, which is a contradiction. Thus, for every variable x_i, we have that $\lambda(\{x_i^T, v\}) = t + 1$ or $\lambda(\{x_i^F, v\}) = t' + 1$ (or both). In particular, this means that if $\lambda(\{x_i, x_i^F\}) = \lambda(\{x_i^F, v\})$, then we set x_i to false, since in this case $\lambda(\{x_i, x_i^T\}) \neq \lambda(\{x_i^T, v\})$.

Now assume for a contradiction that the described assignment is not satisfying. Then there exists a clause c that is not satisfied. Suppose that x_1, x_2, x_3 are three variables that appear in c. Recall that we require $d(c, v) = 2$ and $d(v, c) = \Delta - 1$. The fact that $d(c, v) = 2$ implies that we must have a temporal path consisting of two edges from c to v, such that the two edges have consecutive labels. By construction of G there are three candidates for such a path, one for each literal of c. Assume w.l.o.g. that x_1 appears in c non-negated (the case of a negated appearance of x_1 is symmetrical) and that the temporal path realizing $d(c, v) = 2$ goes through vertex

x_1^T. Let us denote with $t = \lambda(\{x_1^T, v\})$. It follows that $\lambda(\{x_1^T, c\}) = \lambda(\{x_1^T, v\}) - 1 = t-1$. Furthermore, since $d(c, x_1) = 2$ we also have that $\lambda(\{x_1^T, c\}) = \lambda(\{x_1, x_1^T\}) - 1$. Therefore $\lambda(\{x_1, x_1^T\}) = \lambda(\{x_1^T, v\}) = t$. This implies that x_1 is set to TRUE. Let us observe paths from v to c. We know that $d(v, c) = \Delta - 1$. The underlying path of the fastest temporal path from v to c, that goes through x_1^T is the path $P = (v, x_1^T, c)$. Since $\lambda(\{x_1^T, c\}) > \lambda(\{x_1^T, v\})$ we get that the duration of the temporal path (P, λ) is equal to $d(P, \lambda) = (\Delta + t - 1) - t + 1 = \Delta$. This implies that the fastest temporal path from v to c is not (P, λ) and therefore does not pass through x_1^T. Since there are only two other vertices connected to c, we have only two other edges incident to c, that can be used on a fastest temporal path v to c. Suppose now w.l.o.g. that also x_2 appears in c non-negated (the case of a negated appearance of x_2 is symmetrical) and that the temporal path realizing $d(v, c) = \Delta - 1$ goes through vertex x_2^T. Let us denote with $t' = \lambda(\{x_2^T, v\})$. Since the fastest temporal path from v to c is of the duration $\Delta - 1$, and the edge $x_2^T c$ is the only edge incident to vertex c and edge $\{x_2^T, v\}$, it follows that $\lambda(\{x_2^T, c\}) \geq \lambda(\{x_2^T, v\}) - 2 = t' - 2$. Since $d(x_2, v) = 2$ it follows that $\lambda(\{x_2, x_2^T\}) = \lambda(\{x_2^T, v\}) - 1 = t' - 1$. Knowing this and the fact that $d(x_2, c) = 2$, we get that $\lambda(\{x_2^T, c\})$ must be equal to $t' - 2$. Therefore the fastest temporal path from v to c passes through edges $\{x_2^T, v\}$ and $\{x_2^T, c\}$. In the above we have also determined that $\lambda(\{x_2, x_2^T\}) \neq \lambda(\{x_2^T, v\})$, which implies that x_2 is set to FALSE. But now we have that x_1, x_2 both appear in c non-negated, where one of them is TRUE, while the other is FALSE, which implies that the clause c is satisfied, a contradiction.

($\Leftarrow$): Assume that ϕ is satisfiable. Then there exists a satisfying assignment for the variables in ϕ.

We construct a labeling λ as follows.

- All edges incident with a clause vertex c obtain label one.
- If variable x_i is set to TRUE, we set $\lambda(\{x_i^F, v\}) = 3$.
- If variable x_i is set to FALSE, we set $\lambda(\{x_i^T, v\}) = 3$.
- We set the labels of all other edges to two.

For an example of the constructed temporal graph see Figure 6.2. We now verify that all durations are realized.

- For each variable x_i in ϕ we have to check that $d(x_i, v) = 2$ and $d(v, x_i) = \Delta$.

  If x_i is set to TRUE, then there is a temporal path from x_i to v via x_i^F of duration 2, since $\lambda(\{x_i, x_i^F\}) = 2$ and $\lambda(\{x_i^F, v\}) = 3$. For a temporal path from v to x_i, we observe the following. The only possible labels to leave the vertex v are 2 and 3, which take us from v to x_j^T or x_j^F of some variable x_j. The only two edges incident to x_i have labels 2, therefore the fastest path from v to x_i cannot finish before the time $\Delta + 2$. The fastest way to leave v and enter to x_i would then be to leave v at edge $\{x_i^F, v\}$ with label 3, and continue to x_i at time $\Delta + 2$, which gives us the desired duration Δ.

  If x_i is set to FALSE, then, by similar arguing, there is a temporal path from x_i to v via x_i^T of duration 2, and a temporal path from v to x_i, through x_i^F of duration Δ.

- For each clause c in ϕ we have to check that $d(c, v) = 2$ and $d(v, c) = \Delta - 1$:

  Suppose x_i, x_j, x_k appear in c. Since we have a satisfying assignment at least one of the literals in c is set to TRUE and at least one to FALSE. Suppose x_i is the variable of the literal that is TRUE in c, and x_j is the variable of the literal that is FALSE in c. Let x_i appear non-negated in c and is therefore set to TRUE (the case when x_i appears negated in c and is set to FALSE is symmetric). Then there is a temporal path from c to v through x_i^T such that $\lambda(\{x_i^T, c\}) = 1$ and $\lambda(\{x_i^T, v\}) = 2$. Let x_j appear non-negated in c and is therefore set to FALSE (the case when x_j appears negated in c and is set to TRUE is symmetric). Then there is a temporal path from v to c through x_j^T such that $\lambda(\{x_j^T, v\}) = 3$ and $\lambda(\{x_j^T, c\}) = 1$, which results in a temporal path from v to c of duration $\Delta - 1$.

- Let x_i be a variable that appears in clause c. If x_i appears non-negated in c we have to check that $d(c, x_i) = d(c, x_i^F) = 2$ and $d(x_i, c) = d(x_i^F, c) = \Delta$.

  There is a temporal path from c to x_i via x_i^T and also a temporal path from

c to x_i^F via x_i^T such that $\lambda(\{x_i^T, c\}) = 1$ and $\lambda(\{x_i, x_i^T\}) = \lambda(\{x_i^T, x_i^F\}) = 2$, which proves the first equality. There are also the following two temporal paths, first, from x_i to c through x_i^T and second, from x_i^F to c through x_i^T. Both of the temporal paths start on the edge with the label 2, as $\lambda(\{x_i, x_i^T\}) = \lambda(\{x_i^T, x_i^F\}) = 2$ and finish on the edge with label 1, as $\lambda(\{x_i^T, c\}) = 1$.

If x appears negated in c we have to check that $d(c, x_i) = d(c, x_i^T) = 2$ and $d(x_i, c) = d(x_i^T, c) = \Delta$.

There is a temporal path from c to x via x^F and also a temporal path from c to x^T via x^F such that $\lambda(\{c, x^F\}) = 1$ and $\lambda(\{x, x^F\}) = \lambda(\{x^T, x^F\}) = 2$, which proves the first inequality. There are also the following two temporal paths, first, from x_i to c through x_i^F and second, from x_i^T to c through x_i^F. Both of the temporal paths start on the edge with the label 2, as $\lambda(\{x_i, x_i^F\}) = \lambda(\{x_i^T, x_i^F\}) = 2$ and finish on the edge with label 1, as $\lambda(\{x_i^F, c\}) = 1$. Which proves the second equality.

- Let x_i be a variable that does *not* appear in clause c, then we have to check that first, $d(c, x_i^T) = d(c, x_i^F) = 2$, second, $d(x_i^T, c) = d(x_i^F, c) = \Delta$, third, $d(c, x_i) = \Delta + 2$, and fourth $d(x_i, c) = 2\Delta$.

  Let x_j be a variable that appears non-negated in c (the case where x_j appears negated is symmetric). Then there is a temporal path from c to x_i^T via x_j^T and also a temporal path from c to x_i^F via x_j^T such that $\lambda(\{x_j^T, c\}) = 1$ and $\lambda(\{x_j^T, x_i^T\}) = \lambda(\{x_j^T, x_i^F\}) = 2$, which proves the first equality. Using the same temporal path in the opposite direction, i. e., first the edge $x_j^T c$ and then one of the edges $\{x_j^T, x_i^F\}$ or $\{x_j^T, x_i^T\}$ at times 2 and $\Delta + 1$, respectively, yields the second equality. For a temporal path from c to x_i, we traverse the following three edges $\{x_j^T, c\}$, $\{x_j^T, x_i^F\}$, and $\{x_i^F, x_i\}$, with labels 1, 2, and 2 respectively (i. e., the path traverses them at time $1, 2$ and $\Delta + 2$, respectively), which proves the third equality. Now for the case of a temporal path from x_i to c, we use the same three edges but in the opposite direction, namely $\{x_i^F, x_i\}$, $\{x_j^T, x_i^F\}$, and $\{x_j^T, c\}$, again at times 2, $\Delta + 2$, and $2\Delta + 1$, respectively, which proves the last equality. Note that all of the above temporal paths are also the shortest possible, and since the labels of the first and last edges (of these paths) are unique, it follows that we cannot find faster temporal paths.

- For each pair of variables $x_i \neq x_j$ in ϕ we have to check that $d(x_i, x_j) = 2\Delta + 1$ and $d(x_i, x_j^T) = d(x_i, x_j^F) = \Delta + 1$.

  There is a path from x_i to x_j that passes first through one of the vertices x_i^T or x_i^F, and then through one of the vertices x_j^T or x_j^F. This temporal path is of length 3, where all of the edges have label 2, which proves the first equality. Now, a temporal path from x_i to x_j^T (resp. x_j^F), passes through one of the vertices x_i^T or x_i^F. This path is of length two, where all of the edges have label 2, which proves the second equality. Note that all of the above temporal paths are also the shortest possible, and since the labels of the first and last edges (of these paths) are unique, it follows that we cannot find faster temporal paths.

- For each pair of clauses $c_i \neq c_j$ in ϕ we have to check that $d(c_i, c_j) = \Delta + 1$.

  Let x_k be a variable that appears non-negated in c_i and x_ℓ the variable that appears non-negated in c_j (all other cases are symmetric). There is a path of length three from c_i to c_j that passes first through vertex x_k^T and then through vertex x_ℓ^T. Therefore the temporal path from c_i to c_j uses the edges $\{x_k^T, c_i\}$, $\{x_\ell^T, c_j\}$, and $\{x_k^T, x_\ell^T\}$, with labels 1, 2, and 1 (at times 1, 2, and $\Delta + 1$), respectively, which proves the desired equality. Note also that this is the shortest path between c_i and c_j, and that the first and the last edge must have the label 1, therefore it follows that this is the fastest temporal path.

Lastly, observe that the above constructed labeling λ uses values $\{1, 2, 3\} \subseteq [\Delta]$, therefore $\Delta \geq 3$. $\square$

When investigating the parameterized computational hardness of Simple TGR with respect to structural parameters of the underlying graph, it turns out that Simple TGR is W[1]-hard when parameterized by the feedback vertex number of the underlying graph. The *feedback vertex number* of a graph G is the cardinality of a minimum vertex set $X \subseteq V(G)$ such that $G - X$ is a forest. The set X is called a *feedback vertex set*.

Theorem 6.2.2. *Simple TGR is W[1]-hard when parameterized by the feedback vertex number of the underlying graph.*

The proof of this result can be found in [89].

6.3 Algorithms for Simple TGR

In this section, we provide several algorithms for Simple TGR. By Theorem 6.2.1 we have that Simple TGR is NP-hard in general, hence we start by identifying restricted cases where we can solve the problem in polynomial time. We first show in Section 6.3.1 that if the underlying graph G of an instance (D, Δ) of Simple TGR is a tree, then we can determine desired Δ-periodic labeling λ of G in polynomial time. In Section 6.3.2 we generalize this result. We show that Simple TGR is fixed-parameter tractable when parameterized by the feedback edge number of the underlying graph. Note that our parameterized hardness result (Theorem 6.2.2) implies that we presumably cannot replace the feedback edge number with the smaller parameter feedback vertex number, or any other parameter that is smaller than the feedback vertex number, such as, e.g. the treewidth.

6.3.1 Polynomial-time algorithm for trees

We now provide a polynomial-time algorithm for Simple TGR when the underlying graph is a tree. The main idea behind the algorithm utilizes the fact that there exists a unique path among each pair of vertices in a tree, therefore we know the exact durations of all paths. We start by determining the structure of the tree from the matrix D, if possible. We then assign a label 1 to an arbitrary edge and from there extend a labeling to a path of length two, that has one labeled and one unlabeled edge. We do this by using the definition of the duration of a path (more precisely, the duration of a path of length two is the value of the last label minus the value of the first label plus one). We repeat this process until all edges have been labeled. At the end we check if the labeled temporal graph satisfies the fastest path durations from the matrix D.

Theorem 6.3.1. *Simple TGR can be solved in polynomial time on trees.*

Proof. Let D be an input matrix of Simple TGR of dimension $n \times n$. Let us fix the vertices of the corresponding graph G of D as $v_1, v_2, \ldots, v_n$, where vertex v_i corresponds to the row and column i of matrix D. At the same time, we also check if $D_{i,i} = 0$, for all $i \in [n]$. When G is constructed we run the DFS algorithm on it and check that it has no cycles. If at any step we encounter a problem, our algorithm stops and returns a negative answer.

Having computed G, our algorithm proceeds as follows. We pick an arbitrary edge h and give it label 1, that is, $\lambda(h) = 1$. Now we push all edges adjacent with h into a (initially empty) queue. We repeat the following as long as the queue is not empty:

- Pop edge $e = \{u, v\}$ from the queue. Since e was pushed into the queue, there is an edge e' incident with e that already obtained a label. Let w.l.o.g. $e' = \{v, w\}$. Then we set $\lambda(e) = (\lambda(e') - D_{u,w} + 1) \bmod \Delta$.

- Push all edges incident with e that have not received a label yet into the queue.

When the queue is empty, all edges have received a label. Iterate over all vertex pairs u, v and check whether the fastest path from u to v in (G, λ) has duration $D_{u,v}$. If this check succeeds for all vertex pairs, output yes, otherwise abort.

It is easy to see that the described algorithm runs in polynomial time. In the remainder, we prove that it is correct.

($\Rightarrow$): Since the algorithm checks at the end whether all durations specified in D are realized by the corresponding fastest paths, we clearly face a yes-instance whenever the algorithm outputs a labeling.

($\Leftarrow$): Assume we face a yes-instance, then there exists a labeling $\lambda^\star$ that realizes all durations specified in D. Let $e^\star$ denote the edge initially picked by our algorithm. For all edges e let $\lambda(e) = (\lambda^\star(e) - \lambda^\star(e^\star) + 1) \bmod \Delta$. Clearly, the labeling λ also realizes all durations specified in D since λ is obtained by adding the constant $(1 - \lambda^\star(e^\star))$ modulo Δ to all labels of $\lambda^\star$ which does not change the duration of any temporal path, that is all durations in $(G, \lambda^\star)$ are the same as their counterparts in (G, λ). We claim that our algorithm computes and outputs λ.

We prove that our algorithm computes λ by induction on the distance of the labeled edges to $e^\star$, where the distance of two edges e, e' is defined as the length of a shortest path that uses e as its first edge and e' as its last edge.

Initially, our algorithm labels $e^\star$ with 1, which equals $\lambda(e^\star)$. Now let e be an edge popped off the queue by the algorithm in some iteration, that is on the distance i from $e^\star$. Let e' be the edge incident with e that is on the distance $i-1$ from $e^\star$. Since G is a tree e' has already been considered by the algorithm and thus already has a label. By induction, we have that the algorithm labeled e' with $\lambda(e')$. Assume that $e = \{u, v\}$ and $e' = \{v, w\}$. Since G is a tree there is only one path from u to w in G and it uses edges e and e'. It follows that $\lambda(e') - \lambda(e) + 1 = D_{u,w}$ if $\lambda(e') > \lambda(e)$, and $\lambda(e') - \lambda(e) + \Delta + 1 = D_{u,w}$ otherwise. Our algorithm labels e with $(\lambda(e') - D_{u,w} + 1) \bmod \Delta$. It is straightforward to verify that the label of e computed by the algorithm equals $\lambda(e)$. It follows that the algorithm computes λ. □

6.3.2 FPT-algorithm for feedback edge number

Recall that the main reason, for which Simple TGR is straightforward to solve on trees, is twofold:

- between any pair of vertices v_i and v_j in the tree T, there is a *unique* path P in T from v_i to v_j, and
- in any periodic temporal graph (T, λ, Δ) and any fastest temporal path $P = ((e_1, t_1), \ldots, (e_i, t_i), \ldots, (e_j, t_j), \ldots, (e_{\ell-1}, t_{\ell-1}))$ from v_1 to v_ℓ we have that the sub-path $P' = ((e_i, t_i), \ldots, (e_{j-1}, t_{j-1}))$ is also a fastest temporal path from v_i to v_j.

However, these two nice properties do not hold when the underlying graph is not a tree. For example, in Figure 6.3, the fastest temporal path from u to v is $P_{u,v}$ (depicted in blue) goes through w, however, the sub-path of $P_{u,v}$ that stops at w is not the fastest temporal path from u to w. The fastest temporal path from u to w consists only of the single edge uw (with label 9 and duration 1, depicted in red).

Nevertheless, we prove in this section that we can still solve Simple TGR efficiently if the underlying graph is similar to a tree; more specifically we show

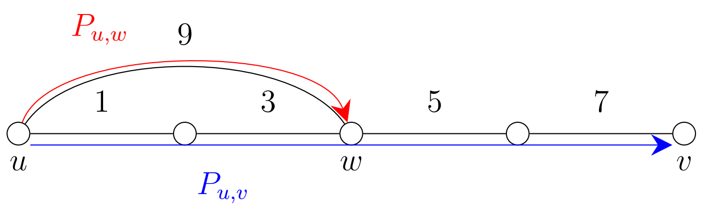


Figure 6.3: An example of a temporal graph (with $\Delta \geq 9$), where the fastest temporal path $P_{u,v}$ (in blue) from u to v is of duration 7, while the fastest temporal path $P_{u,w}$ (in red) from u to a vertex w, that is on a path $P_{u,v}$, is of duration 1 and is not a subpath of $P_{u,v}$.

the following result, which turns out to be non-trivial.

Theorem 6.3.2. *Simple TGR is in FPT when parameterized by the feedback edge number of the underlying graph.*

From Theorem 6.2.2 and Theorem 6.3.2 we immediately get the following, which is the main result of the chapter.

Corollary 6.3.3. *Simple TGR is:*

- *in FPT when parameterized by the* feedback edge number *or any larger parameter, such as the* maximum leaf number*.*
- *W[1]-hard when parameterized by the* feedback vertex number *or any smaller parameter, such as:* treewidth*,* degeneracy*,* cliquewidth*,* distance to chordal graphs*, and* distance to outerplanar graphs*.*

Before presenting the structure of our algorithm for Theorem 6.3.2, observe that, in a static graph, the number of paths between two vertices can be upper-bounded by a function $f(k)$ of the feedback edge number k of the graph [27]. This is true as any such path can traverse $0, 1, 2, \ldots k$ feedback edges in different order. Therefore, for any fixed pair of vertices u and v, we can "guess" the edges of the fastest temporal path from u to v (by guess we mean enumerate and test all possibilities). However, for an FPT algorithm with respect to k, we cannot afford to guess the edges of the fastest temporal path for each of the $O(n^2)$ pairs of vertices. To overcome this difficulty, our algorithm follows this high-level strategy:

- We identify a small number $f(k)$ of "important vertices".

- For each pair u, v of important vertices, we guess the edges of the fastest temporal path from u to v (and from v to u).
- From these guesses we can still not deduce the edges of the fastest temporal paths between many pairs of non-important vertices. However, as we prove, it suffices to guess only a small number of specific auxiliary structures (to be defined later).
- From these guesses we deduce fixed relationships between the labels of most of the edges of the graph.
- For all the edges, for which we have not deduced a label yet, we introduce a *variable*. With all these variables, we build an Integer Linear Program (ILP). Among the constraints in this ILP, we have that, for each of the $O(n^2)$ pairs of vertices u, v in the graph, the duration of one specific temporal path from u to v (according to our guesses) is *equal* to the desired duration $D_{u,v}$, while the duration of each of the other temporal path from u to v is *at least* $D_{u,v}$.
- Each specific configuration of fastest temporal paths among all pairs of vertices corresponds to a specific ILP instance. By exhaustively trying all possible fastest temporal paths configurations it follows that our instance of Simple TGR has a solution if and only if at least one of these ILPs has a feasible solution. As each ILP can be solved in FPT time with respect to k by Lenstra's Theorem [92] (the number of variables is upper bounded by a function of k), we obtain our FPT algorithm for Simple TGR with respect to k.

For the remainder of this section, we fix the following notation. Let D be the input matrix of Simple TGR, i. e., the matrix of the fastest temporal paths between all pairs of n vertices, and let G be its underlying graph, on n vertices and m edges. With F we denote a minimum feedback edge set of G, and with k the feedback edge number of G. We are now ready to present our FPT algorithm. For easier readability, we split the description and analysis of the algorithm into five subsections. We start with a preprocessing procedure for graph G, where we define a set of interesting vertices which then allows us to guess the desired structures.

Next, we introduce some extra properties of our problem, that we then use to create ILP instances and their constraints. At the end, we present how to solve all instances and produce the desired labeling λ of G, if possible.

Preprocessing of the input

From the underlying graph G of D we extract a (connected) graph G' by iteratively removing vertices of degree one from G, and denote with

$$Z = V(G) \setminus V(G').$$

Then we determine a minimum feedback edge set F of G'. Note that F is also a minimum feedback edge set of G. Lastly, we determine sets U, of *vertices of interest*, and U^* of the neighbors of vertices of interest, in the following way. Let T be a spanning tree of G', with F being the corresponding feedback edge set of G'. Let $V_1 \subseteq V(G')$ be the set of leaves in the spanning tree T, $V_2 \subseteq V(G')$ be the set of vertices of degree two in T, that are incident to at least one edge in F, and let $V_3 \subseteq V(G')$ be the set of vertices of degree at least 3 in T. Then $|V_1| + |V_2| \leq 2k$, since every leaf in T and every vertex in V_2 is incident to at least one edge in F, and $|V_3| \leq |V_1|$ by the properties of trees. We denote with

$$U = V_1 \cup V_2 \cup V_3$$

the set of vertices of interest. It follows that $|U| \leq 4k$. We set U^* to be the set of vertices in $V(G') \setminus U$ that are neighbors of vertices in U, i. e.,

$$U^* = \{v \in V(G') \setminus U : \exists u \in U, v \in N(u)\}.$$

Again, using the tree structure, we get that for any $u \in U$ its neighborhood is of size $|N(u)| \in O(k)$, since every neighbor of u is the first vertex of a (unique) path to another vertex in U. It follows that $|U^*| \in O(k^2)$.

From the construction of Z (i. e., by exhaustively removing vertices of degree one from G), it follows that $G[Z]$ (the graph induced in G by Z) is a forest, i. e.,

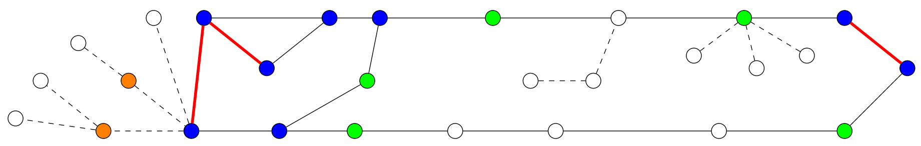


Figure 6.4: An example of a graph with its important vertices: U (in blue), U^* (in green) and Z^* (in orange). Corresponding feedback edges are marked with a thick red line, while dashed edges represent the edges (and vertices) "removed" from G' at the initial step.

consists of disjoint trees. Each of these trees has a unique neighbor v in G'. Denote by T_v the tree obtained by considering such a vertex v and all the trees from $G[Z]$ that are incident to v in G. We then refer to v as the *clip vertex* of the tree T_v. In the case where v is a vertex of interest, we define also the set Z^*_v of *representative vertices* of T_v, as follows. We first create an empty set C_w for every vertex w that is a neighbor of v in G'. We then iterate through every vertex r that is in the first layer of the tree T_v (i. e., vertex that is a child of the root v in the tree T_v), check the matrix D and find the vertex $w \in N_{G'}(v)$ that is on the smallest duration from r. In other words, for an $r \in N_{T_v}(v)$ we find $w \in N_{G'}(v)$ such that $D_{r,w} \leq D_{r,w'}$ for all $w' \in N_{G'}(v)$. We add vertex r to C_w. In the case when there exists also another vertex $w' \in N_{G'}(v)$ for which $D_{r,w'} = D_{r,w}$, we add r also to the set $C_{w'}$. In fact, in this case, $C_{w'} = C_w$. At the end we create $|N_{G'}(v)| \in O(k)$ sets C_w, whose union contains all children of v in T_v. For every two sets C_w and $C_{w'}$, where $w, w' \in N_{G'}(v)$, we have that either $C_w = C_{w'}$, or $C_w \cap C_{w'} = \emptyset$. We interpret each of these sets $\{C_w : w \in N_{G'}(v)\}$ as an *equivalence class* of the neighbors of v in the tree T_v. Now, from each equivalence class C_w, we choose an arbitrary vertex $r_w \in C_w$ and put it into the set Z^*_v. We repeat the above procedure for all trees T_u with the clip vertex u from U, and define Z^* as

$$Z^* = \bigcup_{v \in U} Z^*_v. \tag{6.1}$$

Since $|U| \in O(k)$ and for each $u \in U$ it holds $|N_{G'}(u)| \in O(k)$, we get that $|Z^*| \in O(k^2)$. Finally, the set of *important vertices* is defined as the set $U \cup U^* \cup Z^*$. For an illustration see Figure 6.4. Note that determining sets U, U^* and Z^* takes linear time.

Recall that a labeling λ of G satisfies D if the duration of a fastest temporal path from each vertex v_i to each other vertex v_j equals D_{v_i,v_j}. In order to find a labeling that satisfies this property we split our analysis into nine cases. We consider the fastest temporal paths where the starting vertex is in one of the sets $U, V(G') \setminus U, Z$, and similarly the destination vertex is in one of the sets $U, V(G') \setminus U, Z$. In each of these cases, we guess the underlying path P that at least one fastest temporal path from the vertex v_i to v_j follows, which results in one equality constraint for the labels on the path P. For all other temporal paths from v_i to v_j we know that they cannot be faster, so we introduce inequality constraints for them. This results in producing $f(k) \cdot |D|^{O(1)}$ constraints, where $|D| = n^2$. Note that we have to do this while keeping the total number of variables upper-bounded by some function in k.

For an easier understanding and analysis of the algorithm, we give the following definition.

Definition 6.3.4. *Let $U \subseteq V(G')$ be a set of vertices of interest and let $u, v \in U$. A path $P = (u = v_1, v_2, \ldots, v_p = v)$ with at least two edges in graph G', where all inner vertices are not in U, i. e., $v_i \notin U$ for all $i \in \{2, 3, \ldots, p-1\}$, is called a* segment *from u to v, which we denote as $S_{u,v}$.*

Note from Definition 6.3.4 that $S_{u,v} \neq S_{v,u}$ since we consider paths to be directed. It is also worth emphasizing that $S_{v,u}$ is essentially the reverse path of $S_{u,v}$. Furthermore, it's important to observe that a temporal path in G' between two vertices of interest is either a segment or consists of a sequence of segments. Moreover, any inner vertex v_i in the segment $S_{u,v}$ ($v_i \in S_{u,v} \setminus \{u, v\}$) is part of precisely two segments: $S_{u,v}$ and $S_{v,u}$. Given that we have at most $4k$ interesting vertices in G', we can deduce the following crucial result.

Corollary 6.3.5. *There are $O(k^2)$ segments in G'.*

Guessing necessary structures

Once the sets U, U^* and Z^* are determined, we are ready to start guessing the necessary structures. Note that whenever we say that we guess the fastest temporal path between two vertices, we mean that we guess the underlying path of a representative fastest temporal path between those two vertices. To describe the guesses, we introduce the following notation. Let u, v, x be three vertices in G'. We write $u \rightsquigarrow x \rightarrow v$ to denote a temporal path from u to v that passes through x, and then goes directly to v (via one edge or a unique path in G'). In other words, if the fastest path between two vertices is not uniquely determined we denote it by $\rightsquigarrow$, while if it is unique we denote it by $\rightarrow$. We guess the following paths.

G-1. The fastest temporal paths between all pairs of vertices of U. For a pair u, v of vertices in U, there are $k^{O(k)}$ possible paths in G' between them. Therefore, we have to try all $k^{O(k)}$ possible paths, where at least one of them will be a fastest temporal path from u to v, respecting the values from D. Repeating this procedure for all pairs of vertices $u, v \in U$ we get $k^{O(k^3)}$ different variations of the fastest temporal paths between all pairs of vertices in U.

G-2. The fastest temporal paths between all pairs of vertices in Z^*, which by similar arguing as for vertices in U, gives us $k^{O(k^5)}$ guesses.

G-3. The fastest temporal paths between all pairs of vertices in U^*. This gives us $k^{O(k^5)}$ guesses.

G-4. The fastest temporal paths from vertices of U to vertices in U^*, and vice versa, the fastest temporal paths from vertices in U^* to vertices in U. This gives us $k^{O(k^4)}$ guesses.

G-5. The fastest temporal paths from vertices of U to vertices in Z^*, and vice versa. This gives us $k^{O(k^4)}$ guesses.

G-6. The fastest temporal paths from vertices of U^* to vertices in Z^*, and vice versa. This gives us $k^{O(k^5)}$ guesses.

With the information provided by the described guesses, we are still not able to determine all fastest paths. For example consider the case depicted in Figure 6.5. Therefore, we introduce additional guesses that provide us with sufficient information to determine all fastest paths. We guess the following structures.

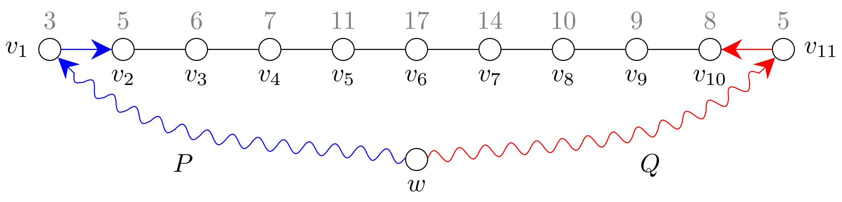


Figure 6.5: In the above graph vertices v_1, v_{11}, w are in U, while v_2, v_{10} are in U^*. Numbers above all v_i represent the values of the fastest temporal paths from w to each of them (i. e., the entries in the w-th row of matrix D). From the basic guesses, we know the fastest temporal path P from w to v_2 (depicted in blue) and the fastest temporal path Q from w to v_{10}. From the values of durations from w to each v_i we cannot determine the fastest paths from w to all v_i. More precisely, we know that w reaches v_2, v_3, v_4, v_5 (resp. v_{10}, v_9, v_9, v_7) by first using the path P (resp. Q) and then proceeding through the vertices, but we do not know how w reaches v_6 the fastest. Therefore we have to introduce some more guesses.

G-7. **Inner segment guess I**. Let $S_{u,v} = (u = v_1, v_2, \ldots, v_p = v)$ and $S_{w,z} = (w = z_1, z_2, \ldots, z_r = z)$ be two segments in G'. We want to guess the fastest temporal path $v_2 \to u \rightsquigarrow w \to z_2$. We repeat this procedure for all pairs of segments. Since there are $O(k^2)$ segments in G', there are $k^{O(k^5)}$ possible paths of this form.

Recall that $S_{u,v} \neq S_{v,u}$ for every $u, v \in U$. Furthermore note that we did not assume that $\{u, v\} \cap \{w, z\} = \emptyset$. Therefore, by repeatedly making the above guesses, we also guess the following fastest temporal paths: $v_2 \to u \rightsquigarrow z \to z_{r-1}$, $v_2 \to u \rightsquigarrow v \to v_{p-1}$, $v_{p-1} \to v \rightsquigarrow w \to z_2$, $v_{p-1} \to v \rightsquigarrow z \to z_{r-1}$, and $v_{p-1} \to v \rightsquigarrow u \to v_2$. For an example see Figure 6.6a.

G-8. **Inner segment guess II**. Let $S_{u,v} = (u = v_1, v_2, \ldots, v_p = v)$ be a segment in G', and let $w \in U \cup Z^*$. We want to guess the following fastest temporal paths $w \rightsquigarrow u \to v_2$, $w \rightsquigarrow v \to v_{p-1} \to \cdots \to v_2$, and $v_2 \to u \rightsquigarrow w$, $v_2 \to v_3 \to \cdots v \rightsquigarrow w$.

For fixed $S_{u,v}$ and $w \in U \cup Z^*$ we have $k^{O(k)}$ different possible such paths, therefore we make $k^{O(k^5)}$ guesses for these paths. For an example see Figure 6.6b.

G-9. **Split vertex guess I**. Let $S_{u,v} = (u = v_1, v_2, \ldots, v_p = v)$ be a segment in G', and let us fix a vertex $v_i \in S_{u,v} \setminus \{u, v\}$. In the case when $S_{u,v}$ is of length 4, the fixed vertex v_i is the middle vertex, else we fix an arbitrary vertex

$v_i \in S_{u,v} \setminus \{u, v\}$. Let $S_{w,z} = (w = z_1, z_2, \dots, z_r = z)$ be another segment in G'. We want to determine the fastest paths from v_i to all inner vertices of $S_{w,z}$. We do this by inspecting the values in matrix D from v_i to inner vertices of $S_{w,z}$. We split the analysis into two cases.

(a) There is a single vertex $z_j \in S_{w,z}$ for which the duration from v_i is the biggest. More specifically, $z_j \in S_{w,z} \setminus \{w, z\}$ is the vertex with the biggest value D_{v_i,z_j}. We call this vertex a *split vertex of v_i in the segment S_{wz}*. Then it holds that $D_{v_i,z_2} < D_{v_i,z_3} < \cdots < D_{v_i,z_j}$ and $D_{v_i,z_{r-1}} < D_{v_i,z_{r-2}} < \cdots < D_{v_i,z_j}$. From this it follows that the fastest temporal paths from v_i to $z_2, z_3, \dots, z_{j-1}$ go through w, and the fastest temporal paths from v_i to $z_{r-1}, z_{r-2}, \dots, z_{j+1}$ go through z. We now want to guess which vertex w or z is on a fastest temporal path from v_i to z_j. Similarly, all fastest temporal paths starting at v_i have to go either through u or through v, which also gives us two extra guesses for the fastest temporal path from v_i to z_j. Therefore, all together we have 4 possibilities on how the fastest temporal path from v_i to z_j starts and ends. Besides that, we want to guess also how the fastest temporal paths from v_i to z_{j-1}, z_{j+1} start and end. Note that one of these is the subpath of the fastest temporal path from v_i to z_j, and the ending part is uniquely determined for both of them, i. e., to reach z_{j-1} the fastest temporal path travels through w, and to reach z_{j+1} the fastest temporal path travels through z. Therefore we have to determine only how the path starts, namely if it travels through u or v. This introduces two extra guesses. For a fixed $S_{u,v}, v_i$ and $S_{w,z}$ we find the vertex z_j in polynomial time, or determine that z_j does not exist. We then make four guesses where we determine how the fastest temporal path from v_i to z_j passes through vertices u, v and w, z and for each of them two extra guesses to determine the fastest temporal path from v_i to z_{j-1} and from v_i to z_{j+1}. We repeat this procedure for all pairs of segments, which results in producing $k^{O(k^5)}$ new guesses. Note, $v_i \in S_{u,v}$ is fixed when calculating the split vertex for all other segments $S_{w,z}$.

(b) There are two vertices $z_j, z_{j+1} \in S_{w,z}$ for which the duration from v_i is the biggest. More specifically, $z_j, z_{j+1} \in S_{w,z} \setminus \{w, z\}$ are the vertices with the biggest value $D_{v_i,z_j} = D_{v_i,z_{j+1}}$. Then it holds that $D_{v_i,z_2} < D_{v_i,z_3} < \dots < D_{v_i,z_j} = D_{v_i,z_{j+1}} > D_{v_i,z_{j+2}} > \dots > D_{v_i,z_{r-1}}$. From this it follows that the fastest temporal paths from v_i to $z_2, z_3, \dots, z_j$ go through w, and the fastest temporal paths from v_i to $z_{r-1}, z_{r-2}, \dots, z_{j+1}$ go through z. In this case, we only need to guess the following two fastest temporal paths $v_i \rightsquigarrow w \to z_2$ and $v_i \rightsquigarrow z \to z_{r-1}$. Each of these paths we then uniquely extend along the segment $S_{w,z}$ up to the vertex z_j, resp. z_{j+1}, which give us fastest temporal paths from v_i to z_j and from v_i to z_{j+1}. In this case we introduce only two more guesses. We repeat this procedure for all pairs of segments. which results in creating $k^{O(k^5)}$ new guesses.

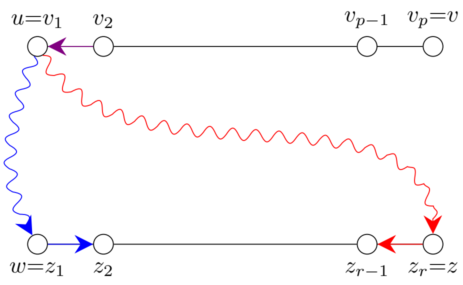
For an example see Figure 6.6c.

G-10. **Split vertex guess II**. Let $w \in U \cup Z^*$, and let $S_{u,v} = (u = v_1, v_2, \dots, v_p = v)$ be a segment in G'. We want to determine the split vertex of w in $S_{u,v}$, and guess the fastest temporal path that reaches it. We again have two cases, first one where v_i is a unique vertex in $S_{u,v}$ that is furthest away from w, and the second one where v_i, v_{i+1} are two incident vertices in $S_{u,v}$, that are furthest away from w. All together we make two guesses for each pair $w, S_{u,v}$. We repeat this for all vertices in $U \cup Z^*$, and all segments, which produces $k^{O(k^5)}$ new guesses. For an example see Figure 6.6d.

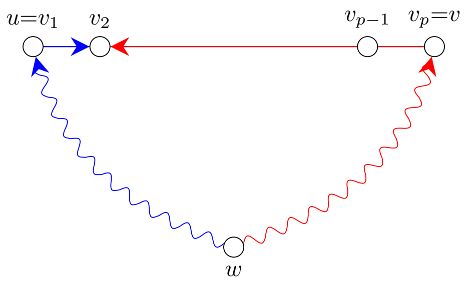
There are two more guesses G-11 and G-12 that we make during the creation of the ILP instances, we explain these guesses in detail in Section 6.3.2. We will prove that, for all guesses G-1 to G-12, there are in total at most $f(k)$ possible choices, and for each one of them we create an ILP with at most $f(k)$ variables and at most $f(k) \cdot |D|^{O(1)}$ constraints. Each of these ILPs can be solved in FPT time by Lenstra's Theorem [92].

Properties of Simple TGR

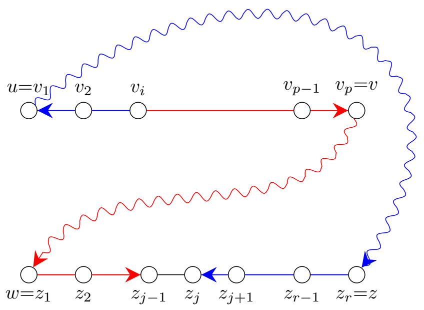
In this section we study the properties of our problem, which then help us create constraints of our ILP instances. Recall that with G we denote our underlying



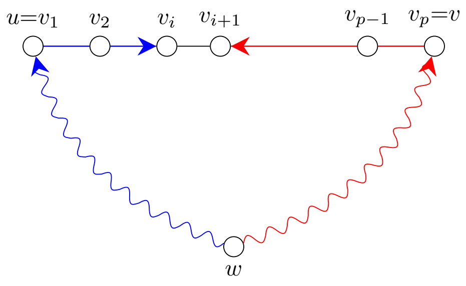
(a) Example of an Inner segment guess I (G-7), where we guessed the fastest temporal paths of the form $v_2 \to u \rightsquigarrow w \to z_2$ (in blue) and $v_2 \to u \rightsquigarrow z \to z_{r-1}$ (in red).



(b) Example of an Inner segment guess II (G-8), where we guessed the fastest temporal paths of the form $w \rightsquigarrow u \to v_2$ (in blue) and $w \rightsquigarrow v \to v_{p-1} \to v_2$ (in red).



(c) Example of a Split vertex guess I (G-9), where, for a fixed vertex $v_i \in S_{u,v}$, we calculated its corresponding split vertex $z_j \in S_{w,z}$, and guessed the fastest paths of the form $v_i \to v_{i-1} \to \cdots \to u \rightsquigarrow z \to z_{r-1} \cdots \to z_j$ (in blue) and $v_i \to v_{i+1} \to \cdots \to v \rightsquigarrow w \to z_2 \to \cdots \to z_{j-1}$ (in red).



(d) Example of a Split vertex guess II (G-10), where, for a vertex of interest w, we calculated its corresponding split vertex $v_i \in S_{u,v}$, and guessed the fastest paths of the form $w \rightsquigarrow u \to v_2 \to \cdots \to v_i$ (in blue) and $w \rightsquigarrow v \to v_{p-1} \to \cdots \to v_{i+1}$ (in red).

Figure 6.6: Illustration of the guesses G-7, G-8, G-9, and G-10.

graph of D. We want to determine labeling λ of each edge of G. We start with empty labeling of edges and try to specify each one of them. Note, that this does not necessarily mean that we assign numbers to the labels, but we might specify labels as variables or functions of other labels. We say that the label of an edge f is *determined with respect to* the label of the edge e, if we have determined $\lambda(f)$ as a function of $\lambda(e)$.

We first start with defining certain notions, that will be of use when solving the problem.

Definition 6.3.6 (Travel delays)**.** *Let (G, λ) be a temporal graph satisfying conditions of* Simple TGR*. Let $e_1 = uv$ and $e_2 = vz$ be two incident edges in G with*

$e_1 \cap e_2 = v$. *We define the* travel delay *from u to z at vertex v, denoted with τ_v^{uz}, as the difference of the labels of e_2 and e_1, where we subtract the value of the label of e_1 from the label of e_2, modulo Δ, if the labels are different, or we set it to Δ. More specifically:*

- *if $\lambda(e_1) \neq \lambda(e_2)$ we have*

$$\tau_v^{uz} \equiv \lambda(e_2) - \lambda(e_1) \pmod{\Delta}, \tag{6.2}$$

- *if $\lambda(e_1) = \lambda(e_2)$ we have*

$$\tau_v^{uz} = \Delta.$$

Intuitively, the value of τ_v^{uz} represents how long a temporal path waits at vertex v when first taking edge $e_1 = uv$ and then edge $e_2 = vz$.

From the above definition and the definition of the duration of the temporal path P, we get the following two observations.

Observation 6.3.7. *Let $P = (v_0, v_1, \ldots, v_p)$ be the underlying path of the temporal path (P, λ) from v_0 to v_p. Then $d(P, \lambda) = \sum_{i=1}^{p-1} \tau_{v_i}^{v_{i-1}v_i} + 1$.*

Proof. For the simplicity of the proof denote $t_i = \lambda(v_{i-1}v_i)$, and suppose that $t_i \leq t_{i+1}$, for all $i \in \{1, 2, 3, \ldots, p\}$. Then

$$\begin{aligned}
\sum_{i=1}^{p-1} \tau_{v_i}^{v_{i-1}v_i} + 1 &= \sum_{i=1}^{p-1} (t_{i+1} - t_i) + 1 \\
&= (t_2 - t_1) + (t_3 - t_2) + \cdots + (t_p - t_{p-1}) + 1 \\
&= t_{p-1} - t_1 + 1 \\
&= d(P, \lambda)
\end{aligned}$$

Now in the case when $t_i > t_{i+1}$ we get that $\tau_{v_i}^{v_{i-1}v_{i+1}} = \Delta + t_{i+1} - t_i$. In the end, this still results in the correct duration as the last time we traverse the path P is not exactly t_p but $k\lambda + t_p$, for some k. □

We also get the following.

Observation 6.3.8. *Let (G, λ) be a temporal graph satisfying conditions of the* Simple *TGR problem. For any two incident edges $e_1 = uv$ and $e_2 = vz$ on vertices $u, v, z \in V$, with $e_1 \cap e_2 = v$, we have $\tau_v^{zu} = \Delta - \tau_v^{uz} \pmod{\Delta}$.*

Proof. Let $e_1 = uv$ and $e_2 = vz$ be two edges in G for which $e_1 \cap e_2 = v$. By the definition $\tau_v^{uz} \equiv \lambda(e_2) - \lambda(e_1) \pmod{\Delta}$ and $\tau_v^{zu} \equiv \lambda(e_1) - \lambda(e_2) \pmod{\Delta}$. Summing now both equations we get $\tau_v^{uz} + \tau_v^{zu} \equiv \lambda(e_2) - \lambda(e_1) + \lambda(e_1) - \lambda(e_2) \pmod{\Delta}$, and therefore $\tau_v^{uz} + \tau_v^{zu} \equiv 0 \pmod{\Delta}$, which is equivalent as saying $\tau_v^{uz} \equiv -\tau_v^{zu} \pmod{\Delta}$ or $\tau_v^{zu} = \Delta - \tau_v^{uz} \pmod{\Delta}$. □

In our analysis, we exploit the following greatly, which is why we state it as an observation.

Observation 6.3.9. *Let P be the underlying path of a fastest temporal path from u to v, where $e_1, e_p \in P$ are its first and last edge, respectively. Then, knowing the label $\lambda(e_1)$ of the first edge and the duration $d(P, \lambda)$ of the temporal path (P, λ), we can uniquely determine the label $\lambda(e_p)$ of the last edge of P. Symmetrically, knowing $\lambda(e_p)$ and $d(P, \lambda)$, we can uniquely determine $\lambda(e_1)$.*

The correctness of the above statement follows directly from **??**. This is because the duration of (P, λ) is calculated as the difference of labels of last and first edge plus 1, where the label of the last edge is considered with some delta periods, i. e., $d(P, \lambda) = p_i\Delta + \lambda(e_p) - \lambda(e_1) + 1$, for some $p_i \geq 0$. Therefore $d(P, \lambda) \pmod{\Delta} \equiv (\lambda(e_p) - \lambda(e_1) + 1) \pmod{\Delta}$. Note that if $\lambda(e_1)$ and $\lambda(e_p)$ are both unknown, then we can determine one with respect to the other.

In the following, we prove that knowing the structure (the underlying path) of a fastest temporal path P from a vertex of interest u to a vertex of interest v, results in determining the labeling of each edge in the fastest temporal path from u to v (with the exception of some constant number of edges), with respect to the label of the first edge. More precisely, if path P from u to v is a segment, then we can determine labels of all edges as a function of the label of the first edge. If P consists of ℓ segments, then we can determine the labels of all but $\ell - 1$ edges as a function of the label of the first edge. For the exact formulation and proofs see Lemmas 6.3.10 and 6.3.11.

Lemma 6.3.10. *Let $u, v \in U$ be two arbitrary vertices of interest and suppose that $P = (u = v_1, v_2, \dots, v_p = v)$, where $p \geq 2$, is a path in G', which is also the underlying path of a fastest temporal path from u to v. Moreover, suppose also that P is a segment. We can determine the labeling λ of every edge in P with respect to the label $\lambda(uv_2)$ of the first edge.*

Proof. We claim that u reaches all of the vertices in P the fastest, when travelling along P (i. e., by using a subpath of P). To prove this suppose for the contradiction that there is a vertex $v_i \in P \setminus \{u, v\}$, that is reached from v on a path different than $P_i = (u, v_2, v_3, \dots, v_i)$ faster than through P_i. Since the only vertices of interest of P are u and v, it follows that all other vertices on P are of degree 2. Then the only way to reach v_i from u, that differs from P, would be to go from u to v using a different path P_2, and then go from v to $v_{p-1}, v_{p-2}, \dots, v_i$. But since P is the fastest temporal path from u to v, we get that $d(P_2) \geq d(P)$ and $d(P_2 \cup (v, v_{p-1}, \dots, v_i)) > d(P) > d(P_i)$.

Now, to determine the labeling λ of the path P we use the property that the fastest temporal path from u to any $v_i \in P$ is a subpath of P. We set the label of the first edge of P to be a constant $c \in [\Delta]$ and use Observation 6.3.9 to label all remaining edges, where the duration from u to v_i equals to D_{u,v_i}. This gives us a unique labeling λ of P where the label of each edge of P is a function of c. □

Lemma 6.3.11. *Let $u, v \in U$ be two arbitrary vertices of interest and suppose that $P = (u = v_1, v_2, \dots, v_p = v)$, where $p \geq 2$, is a path in G', which is also the underlying path of a fastest temporal path from u to v. Let $\ell_{u,v} \geq 1$ be the number of vertices of interest in P different to u, v, namely $\ell_{u,v} = |\{P \setminus \{u, v\}\} \cap U|$. We can determine the labeling λ of all but $\ell_{u,v}$ edges of P, with respect to the label $\lambda(uv_2)$ of the first edge, such that the labeling λ respects the values from D.*

For the proof of the above lemma, we first prove a weaker statement, for which we need to introduce some extra definitions and fix some notations. In the following we only consider *wasteless* temporal paths. We call a temporal path $P = ((e_1, t_1), \dots, (e_k, t_k))$ a *wasteless* temporal path, if for every $i = 1, 2, \dots, k-1$, we have that t_{i+1} is the first time after t_i that the edge e_{i+1} appears.

Let $u, v \in V$, and let $t \in \mathbb{N}$. Given that a temporal path starts within the period $[t, t+\Delta-1]$, we denote with $A_t(u,v)$ the *arrival* of the fastest path in (G,λ) from u to v, and with $A_t(u,v,P)$, the *arrival* along path P in (G,λ) from u to v. Whenever $t = 1$, we may omit the index t, i. e., we may write $A(u,v,P) = A_1(u,v,P)$ and $A(u,v) = A_1(u,v)$.

Suppose now that we know the underlying path $P_{u,v} = (u = v_1, v_2, \dots, v_p = v)$ of the fastest temporal path between vertices of interest u and v in G'. Let $v_i \in U$ with $u \neq v_i \neq v$ be a vertex of interest on the path $P_{u,v}$. Suppose that v_i is reached the fastest from u by a path $P = (u = u_1, u_2, \dots, u_{j-1}, v_i)$. We split the path $P_{u,v}$ into a path $Q = (u = v_1, v_2, \dots, v_i)$ and $R = (v_i, v_{i+1}, \dots, v_p = v)$ (for details see Figure 6.7).

From the above, we get the following assumptions:

1. $d(u,v_i) = d(u,v_i,P) \leq d(u,v_i,Q)$, and

2. $d(u,v_p) = d(u,v_p,Q \cup R) \leq d(u,v_p,P \cup R)$.

In the remainder, we denote with δ_0 the difference $d(u,v_i,Q) - d(u,v_i) \geq 0$. Let $t_{v_2} \in [\Delta]$ be the label of the edge uv_2, and denote by t_{u_2} the appearance of the edge uu_2 within the period $[t_{v_2}, t_{v_2}+\Delta-1]$. Note that $1 \leq t_{v_2} \leq \Delta$ and that $t_{v_2} \leq t_{u_2} \leq 2\Delta$. From Assumption 1 we get

$$\delta_0 = d(u,v_i,Q) - d(u,v_i) = A_{t_{v_2}}(u,v_i,Q) - A_{t_{v_2}}(u,v_i,P) + (t_{u_2} - t_{v_2})$$

and thus

$$A_{t_{v_2}}(u,v_i,P) - A_{t_{v_2}}(u,v_i,Q) = t_{u_2} - (t_{v_2} + \delta_0). \tag{6.3}$$

We use all of the above discussion, to prove the following lemma.

Lemma 6.3.12. *If $t_{u_2} \neq t_{v_2}$, then $\delta_0 \leq \Delta - 2$ and $t_{u_2} \geq t_{v_2} + \delta_0 + 1$.*

Proof. First assume that $\delta_0 \geq \Delta - 1$. Then, it follows by Equation (6.3) that $A_{t_{v_2}}(u,v_i,P) - A_{t_{v_2}}(u,v_i,Q) \leq t_{u_2} - t_{v_2} - \Delta + 1 \leq 0$, and thus $A_{t_{v_2}}(u,v_i,P) \leq A_{t_{v_2}}(u,v_i,Q)$. Therefore, since we can traverse path P from u to v_i by departing at time $t_{u_2} \geq t_{v_2} + 1$ and by arriving no later than traversing path Q, we have

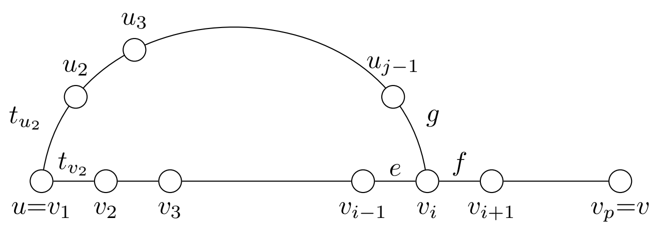


Figure 6.7: An example of the situation in Lemma 6.3.11, where we assume that the fastest temporal path from u to v is $P_{u,v} = (u = v_1, v_2, \ldots v_p)$, and the fastest temporal path from u to v_i is $P = (u, u_2, u_3, \ldots, v_i)$. We denote $Q = (u = v_1, v_2, \ldots, v_i)$ and $R = (v_i, v_{i+1}, \ldots, v_p = v)$.

that $d(u, v_p, P \cup Q) < d(u, v_p, Q \cup R)$, which is a contradiction to the second initial assumption. Therefore $\delta_0 \leq \Delta - 2$.

Now assume that $t_{v_2} + 1 \leq t_{u_2} \leq t_{v_2} + \delta_0$. Then, it follows by Equation (6.3) that $A_{t_{v_2}}(u, v_i, P) \leq A_{t_{v_2}}(u, v_i, Q)$ which is, similarly to the previous case, a contradiction. Therefore $t_{u_2} \geq t_{v_2} + \delta_0 + 1$. □

The next corollary follows immediately from Lemma 6.3.12.

Corollary 6.3.13. *If $t_{u_2} \neq t_{v_2}$, then $1 \leq A_{t_{v_2}}(u, v_i, P) - A_{t_{v_2}}(u, v_i, Q) \leq \Delta - 1 - \delta_0$.*

We are now ready to prove the following result.

Lemma 6.3.14. *$d(u, v_{i-1}, P \cup \{v_i v_{i-1}\}) > d(u, v_{i-1}, Q \setminus \{v_i v_{i-1}\})$.*

Proof. Let $e \in [\Delta]$ be the label of the edge $v_{i-1}v_i$, and let $f \in [e+1, e+\Delta]$ be the time of the first appearance of the edge $v_i v_{i+1}$ after time e. Let $A_{t_{v_2}}(u, v_i, Q) = x\Delta + e$. Then $A_{t_{v_2}}(u, v_{i+1}, Q \cup \{v_i v_{i+1}\}) = x\Delta + f$. Furthermore let g be such that $A_{t_{v_2}}(u, v_i, P) = x\Delta + g$.

Case 1: $t_{u_2} \neq t_{v_2}$. Then Corollary 6.3.13 implies that $e+1 \leq g \leq e+(\Delta-1-\delta_0)$. Assume that $g < f$. Then, we can traverse path P from u to v_i by departing at time $t_{u_2} \geq t_{v_2} + 1$ and by arriving at most at time $x\Delta + f - 1$, and thus $d(u, v_p, P \cup R) < d(u, v_p, Q \cup R)$, which is a contradiction to the second initial assumption. Therefore $g \geq f$. That is,

$$e + 1 \leq f \leq g \leq e + (\Delta - 1 - \delta_0).$$

Consider the path $P^* = P \cup \{v_i v_{i-1}\}$. Assume that we start traversing P^* at time t_{u_2}. Then we arrive at v_i at time $x\Delta + g$, and we continue by traversing edge $v_i v_{i-1}$ at time $(x+1)\Delta + e$. That is, $d(u, v_{i-1}, P^*) = (x+1)\Delta + e - t_{u_2} + 1$.

Now consider the path $Q^* = Q \setminus \{v_i v_{i-1}\}$. Let $h \in [1, \Delta]$ be such that $A_{t_{v_i}}(u, v_{i-1}, Q^*) = x\Delta + e - h$. That is, if we start traversing Q^* at time t_{v_2}, we arrive at v_{i-1} at time $x\Delta + e - h$, i. e., $d(u, v_{i-1}, Q^*) = x\Delta + e - h - t_{v_2} + 1$. Summarizing, we have:

$$\begin{aligned} d(u, v_{i-1}, P^*) - d(u, v_{i-1}, Q^*) &= \Delta + h - (t_{u_2} - t_{v_2}) \\ &\geq \Delta + h - (\Delta - 1) > 0, \end{aligned}$$

which proves the statement of the lemma.

Case 2: $t_{u_2} = t_{v_2}$. Then, it follows by Equation (6.3) that $A_{t_{v_2}}(u, v_i, P) = A_{t_{v_2}}(u, v_i, Q) - \delta_0 \leq A_{t_{v_2}}(u, v_i, Q)$. Therefore $g \leq e$. Similar to Case 1 above, consider the paths $P^* = P \cup \{v_i v_{i-1}\}$ and $Q^* = Q \setminus \{v_i v_{i-1}\}$. Assume that we start traversing P^* at time $t_{u_2} = t_{v_2}$. Then we arrive at v_i at time $x\Delta + g$, and we continue by traversing edge $v_i v_{i-1}$, either at time $(x+1)\Delta + e$ (in the case where $g = e$) or at time $x\Delta + e$ (in the case where $g \neq e$). That is, $d(u, v_{i-1}, P^*) \geq x\Delta + e - t_{v_2} + 1$.

Similarly to Case 1, let $h \in [1, \Delta]$ be such that $A_{t_{v_i}}(u, v_{i-1}, Q^*) = x\Delta + e - h$. That is, if we start traversing Q^* at time t_{v_2}, we arrive at v_{i-1} at time $x\Delta + e - h$, i. e., $d(u, v_{i-1}, Q^*) = x\Delta + e - h - t_{v_1} + 1$. Summarizing, we have:

$$d(u, v_{i-1}, P^*) - d(u, v_{i-1}, Q^*) \geq h \geq 1,$$

which proves the statement of the lemma. $\square$

From the above it follows that if P is a fastest path from u to v, then all vertices of P, with the exception of vertices of interest $v_i \in P \setminus \{u, v\}$, are reached using the same path P. We use this fact in the following proof.

Proof of Lemma 6.3.11. For every vertex of interest $v_i \in U \cap (P \setminus \{u, v\})$ we have two options. First, when the fastest temporal path P' from u to v_i is a subpath of P. In this case, we determine the labeling of P' using Lemma 6.3.10. Second, when the fastest temporal path P' from u to v_i is not a subpath of P. In this case, we know exactly how to label all of the edges of P, with the exception of edges from $v_{i-1}v_i$, that are incident to v_i in P. $\square$

Lemma 6.3.15. *Let $S_{u,v} = (u = v_1, v_2, \dots, v_p = v)$ be a segment in G. If $S_{u,v}$ is of length at least 5 ($p > 5$) then it is impossible for an inner edge $f = v_iv_{i+1}$ from $S_{u,v} \setminus \{u, v\}$ (where f is an edge that is not incident to a vertex from U) to not be a part of any fastest temporal path, of length at least 2 between vertices in $S_{u,v}$. In other words, there must exist a pair $v_j, v_{j'} \in S_{u,v}$ s. t., the fastest temporal path from v_j to $v_{j'}$ passes through f. If $S_{u,v}$ is of length 4 then all temporal paths of length 2 avoid the inner edge f if and only if f has the same label as both of the edges incident to it, while the label of the last remaining edge is determined with respect to $\lambda(f)$.*

Proof. For an easier understanding and better readability, we present the proof for $S_{u,v}$ of length 5. The case where $S_{u,v}$ is longer easily follows from the presented results.

Let $S_{u,v} = (u = v_1, v_2, v_3, v_4, v_5, v_6 = v)$. We distinguish two cases, first when $f = v_2v_3$ (note that the case with $f = v_4v_5$ is symmetrical), and the second when $f = v_3v_4$. Throughout the proof we denote with t_i the label of edge v_iv_{i+1}. Suppose for the contradiction, that none of the fastest temporal paths between vertices of $S_{u,v}$ traverses the edge f.

Case 1: $f = v_2v_3$. Let us observe the case of the fastest temporal paths between v_1 and v_3. Denote $Q = (v_1, v_2, v_3)$ and $P' = (v_3, v_4, v_5, v_6)$. From our proposition, it follows that

- the fastest temporal path P^+ from v_1 to v_3 is of the following form $P^+ = v_1 \rightsquigarrow v_6 \rightarrow v_5 \rightarrow v_4 \rightarrow v_3$, and
- the fastest temporal path P^- from v_3 to v_1 is of the following form $P^- = v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightsquigarrow v_1$.

It follows that $d(v_1, v_3, P^+) \leq d(v_1, v_3, Q)$, and $d(v_3, v_1, P^-) \leq d(v_3, v_1, Q)$. Note that $d(v_1, v_3, P^+) \geq 1 + d(v_6, v_3, P')$, and by the definition $d(v_6, v_3, P') = 1 + (t_4 - t_5)_\Delta + (t_3 - t_4)_\Delta$, where $(t_i - t_j)_\Delta$ denotes the difference of two consecutive labels

t_i, t_j modulo Δ. Similarly it holds for $d(v_1, v_3, P^+)$. Summing now both of the above equations we get

$$\begin{aligned}
d(v_1, v_3, P^+) + d(v_3, v_1, P^-) &\leq d(v_1, v_3, Q) + d(v_3, v_1, Q) \\
1 + d(v_6, v_3, P') + 1 + d(v_3, v_6, P') &\leq d(v_1, v_3, Q) + d(v_3, v_1, Q) \\
3 + (t_4 - t_5)_\Delta + (t_3 - t_4)_\Delta + 1 + (t_4 - t_3)_\Delta + (t_5 - t_4)_\Delta &\leq 1 + (t_2 - t_1)_\Delta + 1 + (t_1 - t_2)_\Delta \\
(t_4 - t_5)_\Delta + (t_5 - t_4)_\Delta + (t_4 - t_3)_\Delta + (t_3 - t_4)_\Delta + 2 &\leq (t_2 - t_1)_\Delta + (t_1 - t_2)_\Delta.
\end{aligned} \tag{6.4}$$

Note that if $t_i \neq t_j$ we get that the sum $(t_i - t_j)_\Delta + (t_j - t_i)_\Delta$ equals exactly Δ, and if $t_i = t_j$ the sum equals 2Δ. This follows from the definition of travel delays at vertices (see Observation 6.3.8). Therefore we get from Equation (6.4), that the right part is at most 2Δ, while the left part is at least $2\Delta + 1$, for any relation of labels $t_1, t_2, \ldots, t_5$, which is a contradiction.

Case 2: $f = v_3v_4$. Here we consider the fastest paths between vertices v_2 and v_4. By similar arguments as above we get

$$(t_5 - t_1)_\Delta + (t_4 - t_5)_\Delta + (t_5 - t_4)_\Delta + (t_1 - t_5)_\Delta + 2 \leq (t_3 - t_2)_\Delta + (t_2 - t_3)_\Delta,$$

which is impossible.

In the case when $S_{u,v}$ is longer, we would get an even bigger number on the left-hand side of Equation (6.4), so we conclude that in all of the above cases, it cannot happen that all fastest paths of length 2, between vertices in $S_{u,v}$, avoid edge f.

Let us observe now the case when $S_{u,v} = (u = v_1, v_2, v_3, v_4, v_5 = v)$ is of length 4. Let $f = v_2v_3$ (the case with $f = v_3v_4$ is symmetrical). Suppose that the fastest temporal paths between v_1 and v_3 do not use the edge f. We denote with R^+ the fastest path from v_1 to v_3, which is of the form $u \rightsquigarrow v \rightarrow v_4 \rightarrow v_3$, and similarly with R^- the fastest path from v_3 to v_1, which is of the form $v_3 \rightarrow v_4 \rightarrow v \rightsquigarrow u$. We denote $R' = (v_3, v_4, v_5)$ and $S = (v_1, v_2, v_3)$. Again we get the following.

$$\begin{aligned}
d(v_1, v_3, R^+) + d(v_3, v_1, R^-) &\leq d(v_1, v_3, S) + d(v_3, v_1, S) \\
1 + d(v_5, v_3, R') + 1 + d(v_3, v_5, R') &\leq d(v_1, v_3, S) + d(v_3, v_1, S) \\
(t_3 - t_4)_\Delta + (t_4 - t_3)_\Delta + 2 &\leq (t_2 - t_1)_\Delta + (t_1 - t_2)_\Delta.
\end{aligned}$$

The only case when the equation has a valid solution is when $t_1 = t_2$ and $t_3 \neq t_4$, as in this case the left hand side evaluates to $\Delta + 2$, while the right side evaluates to 2Δ. Repeating the analysis for the fastest paths between v_2 and v_4, we conclude that the only valid solution is when $t_2 = t_3$ and $t_1 \neq t_4$. Altogether, we get that f is not a part of any fastest path of length 2 in $S_{u,v}$ if and only if the label of edge f is the same as the labels on the edges incident to it, while the last remaining edge has a different label. Note now that the fastest temporal path from v_2 to v_4 must first use the edge uv_2 and finish with the edge v_4v_5, and it has to be of duration D_{v_2,v_4}. Using Lemma 6.3.10 we determine the label of the edge v_4v_5 with respect to $\lambda(f)$. □

We now present some properties involving the vertices from Z, that form the trees in $G[Z]$.

Lemma 6.3.16. *Let $v \in V(G')$ be a clip vertex of the tree T_v in $G[Z \cup \{v\}]$, and let $z \in N_{T_v}(v)$ be an arbitrary child of v in T_v. Among all neighbors of v in G', let w be the one that is on the smallest duration away from z with respect to the values of D. In other words, $w \in N_{G'}(v)$ such that $D_{z,w} \leq D_{z,w'}$ for all $w' \in N_{G'}(v)$. Then, the path $P^* = (z, v, w)$ represents the unique fastest temporal path from z to w. Moreover, we can determine all labels of the tree T_v with respect to the label $\lambda(vw)$.*

Proof. Suppose for contradiction that there exists a path $P^{**} \neq P^*$ from z to w such that $d(P^{**}, \lambda) \leq d(P^*, \lambda)$. By the structure of G, it follows that P^{**} passes through the clip vertex v of T_v (as this is the only neighbor of z in G'), continues through a vertex $w' \in N_{G'}(v) \setminus \{w\}$, and through some other vertices $u_1, u_2, \ldots, u_j$ in G ($j \geq 0$) before finishing in w. Therefore, $P^{**} = (z, v, w', u_1, u_2, \ldots, u_j, w)$. Now, since $D_{z,w} \leq D_{z,w'}$ by assumption, the first part of P^{**} from z to w' takes at least $D_{z,w'}$ time, and thus it takes at least $D_{z,w}$ time. Since $w \neq w'$, we need at least one more time-step (one more edge) to traverse from w' to reach w. Therefore, $d(P^{**}, \lambda) \geq D_{z,w} + 1$ which implies that P^{**} is not the fastest temporal path from z to w. Therefore, the only fastest temporal path from z to w is $P^* = (z, v, w)$.

For the second part, knowing that the duration of P^* is $D_{z,w}$, we can determine the label of the edge zv with respect to the label $\lambda(vw)$ (see Observation 6.3.9). Furthermore, using the algorithm for trees (see Theorem 6.3.1), we can now determine all the labels on the edges of T_v with respect to the same label $\lambda(vw)$. □

Lemma 6.3.17. *Let $x \in V(G')$ be a clip vertex of the tree T_x in $G[Z \cup \{x\}]$, where $x \notin U$. Let v_1 and v_2 be the two neighbors of x in G'. Then the labels of the tree T_x can be determined with respect to $\lambda(v_1x)$ and $\lambda(xv_2)$.*

Proof. First observe that since x is not a vertex of interest it must be a part of some segment $S_{u,w}$, where $u, v \in U$ and $x \neq u \neq v$. Therefore, x is of degree 2 in G'. Let $z \in V(T_x)$ be a child of x in T_x, i. e., a vertex in the first layer of the tree T_x. We observe the values D_{z,v_1}, D_{z,v_2} and distinguish the following cases.

First, $D_{z,v_1} = D_{z,v_2}$ Then, using Lemma 6.3.16 we conclude that the fastest temporal paths from z to v_1 and from z to v_2 are of length two. We know that these two paths consist of the edge zx and xv_1, xv_2, respectively. This allows us to determine the label of the edge zx (and consequently all other edges of T_x) with respect to $\lambda(xv_1)$ and $\lambda(xv_2)$.

Second, $D_{z,v_1} \neq D_{z,v_2}$. Let us denote with $t_1 = \lambda(xv_1), t_2 = \lambda(xv_2)$ and $t_3 = \lambda(zx)$. W.l.o.g. suppose that $\min\{t_1, t_2, t_3\} = t_3$, and that $D_{z,v_1} > D_{z,v_2}$ (the other case is analogous). It follows that $t_1 > t_2$. We want to now prove that the inequality $D_{v_1,z} < D_{v_2,z}$ holds. Suppose for the contradiction that the inequality is false. Then $D_{v_2,z} < D_{v_1,z} \leq (\Delta + t_3 - t_1)$. This implies that the fastest temporal path from v_2 to z cannot use the path (v_1, x, z), and is therefore of form (v_2, x, z). By the definition, the duration of this path is $D_{v_2,z} = \Delta + t_3 - t_2 + 1$. But since $t_1 > t_2$ it follows that $(\Delta + t_3 - t_2) + 1 > (\Delta + t_3 - t_1) + 1$. We also know that $D_{v_1,z} \leq (\Delta + t_3 - t_1) + 1$. This implies that $D_{v_2,z} > D_{v_1,z}$, a contradiction.

Knowing $D_{z,v_1} > D_{z,v_2}$ we can determine the label of edge zx (and consequently all other edges of T_x) with respect to $\lambda(xv_2)$, and similarly knowing $D_{v_1,z} < D_{v_2,z}$ we determine the label of edge zx (and all other edges of T_x) with respect to $\lambda(xv_1)$. □

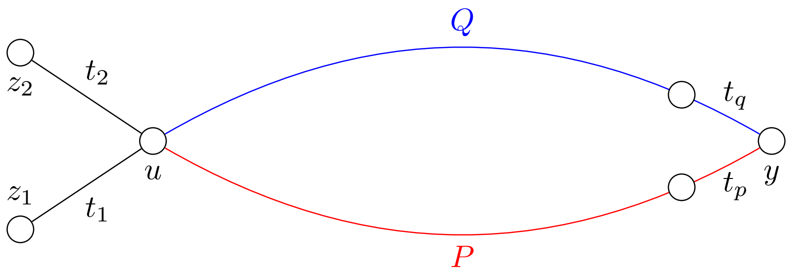


Figure 6.8: An example of the situation in Lemma 6.3.18. We have two paths Q (in blue) and P (in red) from u to y. We assume that the fastest temporal path from z_1 to y is $P_1 = \{z_1u\} \cup P$, and the fastest temporal path from z_2 to y is $Q_2 = \{z_2u\} \cup Q$. We denote with $Q_1 = \{z_1u\} \cup Q$ another temporal path from z_1 to y, with $P_2 = \{z_2u\} \cup P$ another temporal path from z_2 to y. The labels on the edges z_1u, z_2u are t_1, t_2, respectively. Similarly, the labels of the last edges of paths P and Q are t_p, t_q, respectively.

Remember, in the case where the clip vertex u of the tree T_u in $G[Z \cup \{u\}]$ is a vertex of interest, we split the vertices in the first layer of T_u into at most $|N_{G'}(u)|$ equivalence classes (as explained in Section 6.3.2). Let us now show the following important property of these equivalence classes.

Lemma 6.3.18. *Let $u \in V(G')$ be a clip vertex of the tree T_u in $G[Z \cup \{u\}]$, where $u \in U$, and let $z_1, z_2 \in V(T_u)$ be in the same equivalence class of the tree T_u. Then, the fastest temporal paths from z_1 and from z_2 to any other vertex in G' coincide on the edges in G'. Similarly, the fastest temporal paths from any other vertex in G' to z_1 and to z_2 coincide on the edges in G'.*

Proof. Let $y \neq u$ be a vertex in $V(G')$. Denote with P_1 the underlying path of the fastest temporal path from z_1 to y, which consists of the edge z_1u and the path P from u to y. Similarly, let Q_2 be the underlying path of the fastest temporal path from z_2 to y, consisting of the edge z_2u together with the path Q from u to y. Define P_2 as the second path from z_2 to y that first uses the edge z_2u and then the path P. Similarly, Q_1 represents the second path from z_1 to y that first uses the edge z_1u and then the path Q. Our objective is to demonstrate that either $P = Q$ or that $d(P_1, \lambda) = d(Q_1, \lambda)$ (and $d(P_2, \lambda) = d(Q_2, \lambda)$). This implies that z_1 and z_2 use temporal paths that coincide on the vertices of $V(G) \setminus V(T_u)$ to reach y. For an illustration see Figure 6.8.

Let us set the label of the edge z_1u to t_1, the label of z_2u to t_2, the label of the last edge of the path P as t_p and the label of the last edge of the path Q as t_q. By the definition, since P_1 represents the fastest temporal path form z_1 to y we get that $D_{z_1,y} = t_p - t_1 + c_p\Delta$, where $c_p \in \mathbb{N}$. Similarly, for the path Q_2 it holds that $D_{z_2,y} = t_q - t_2 + c_q\Delta$ with $c_q \in \mathbb{N}$. Note that the difference between the first label of P (resp. Q) with t_1 and t_2 is smaller than Δ, or the difference (with at least one t_1, t_2) is Δ if and only if the first label of P and the first label of Q are the same. This observation is crucial in our arguing below.

We want to first show that $c_p = c_q$. Let us assume, for the sake of contradiction, that this is not the case, and suppose that $c_p > c_q$ (the case with $c_q > c_p$ is analogous). Then $c_p \leq c_q + 1$. Now, since z_1 and z_2 are in the same equivalence class and by the definition of the duration of a temporal path we get that $d(P_2, \lambda) = t_p - t_2 + c_p\Delta \leq t_p - t_2 + (c_q - 1)\Delta$. Because Q_2 is the fastest path from z_2 to y we have also that $d(P_2, \lambda) \geq D_{z_2,y}$, which gives us $t_p - t_2 + (c_q - 1)\Delta \geq t_q - t_2 + c_q\Delta$. This is equivalent to $t_p \geq t_q + \Delta$, but since $t_p, t_q \in [\Delta]$ this cannot happen. Therefore, we conclude that $c_p = c_q$.

Now, we want to show also, that $t_p = t_q$. Let us assume, for the sake of contradiction, that this is not the case, and suppose that $t_p > t_q$ (the case with $t_q > t_p$ is analogous). Then the duration of the path Q_1 is $d(Q_1, \lambda) = t_q - t_1 + c_q\Delta$ since $c_q = c_p$. Above we proved that $c_p = c_q$. We also know that $d(Q_1, \lambda) \geq d(P_1, \lambda)$ as P_1 is the fastest path from z_1 to y. All of this results in $t_q - t_1 + c_p\Delta \geq t_p - t_1 + c_p\Delta$ implying $t_q \geq t_p$, a contradiction. Therefore, $t_p = t_q$.

We proved that either P and Q are the same, or if they are different then P_1 and Q_1 are of the same duration and are both fastest paths from z_1 to y (the same holds for z_2).

Proof for the fastest temporal paths in the other direction, namely starting at y and reaching z_1 and z_2, is done analogously. $\square$

Observation 6.3.19. *Let $v \in V(G')$ be a clip vertex of the tree T_v in $G[Z \cup \{v\}]$, $z \in N_{T_v}(v)$ be a child of v in T_v, and let z' be a descendant of z in T_v. Let $x \in V(G) \setminus V(T_v)$ be an arbitrary vertex. Denote by P_z and $P_{z'}$ the underlying paths of the fastest temporal paths from z and from z' to x, respectively, and denote by Q the (unique) path between z and z' in T_v. Then P_z and $P_{z'}$ differ only in the edges of Q.*

The correctness of the above observation is a consequence of Lemma 6.3.18 and of the fact that P_z and $P_{z'}$ leave the tree T_v using the same edge zv.

Adding constraints and variables to the ILP

We start by analyzing the case where we want to determine the labels on fastest temporal paths between vertices of interest. We proceed in the following way. Let $u, v \in U$ be two vertices of interest and let $P_{u,v}$ be the fastest temporal path from u to v. If $P_{u,v}$ is a segment we determine all the labels of edges of $P_{u,v}$, with respect to the label of the first edge (see Lemma 6.3.10). In the case when $P_{u,v}$ is a sequence of ℓ segments, we determine all but $\ell - 1$ labels of edges of $P_{u,v}$, with respect to the label of the first edge (see Lemma 6.3.11). We call these $\ell - 1$ edges, *partially determined* edges. After repeating this step for all pairs of vertices in U, the edges of fastest temporal paths from u to v, where $u, v \in U$, are determined with respect to the label of the first edge of each path or are partially determined. If the fastest temporal path between two vertices $u, v \in U$ is just an edge e, then we treat it as being determined, since it gets assigned a label $\lambda(e)$ with respect to itself. All other edges in G' are called the *not yet determined* edges. Note that the not yet determined edges are exactly the ones that are not a part of any fastest temporal path between any two vertices in U.

Now we want to relate the not yet determined segments with the determined ones. Let $S_{u,v}$ and $S_{w,z}$ be two segments. At the beginning, we have guessed the fastest path from v_i to all vertices in $S_{w,z}$ (see guess G-9). We did this by determining which vertices z_j, z_{j+1} in $S_{w,z}$ are furthest away from v_i (remember we can have the case when $z_j = z_{j+1}$), and then we guessed how the path from v_i leaves the segment $S_{u,v}$ (i. e., either through the vertex u or v), and then how it reaches z_j (in the case when $z_j \neq z_{j+1}$ there is a unique way, when $z_j = z_{j+1}$ we determined which of the vertices w or z is on the fastest path). W.l.o.g. assume that we have guessed that the fastest path from v_i to z_j passes through w and z_{j-1}. Then the fastest temporal path from v_i to z_{j+1} passes through z. And all fastest temporal paths from v_i to any $z_{j'} \in S_{w,z}$ use all of the edges in $S_{w,z}$ with the exception of the edge $z_j z_{j+1}$. Using this information and Observation 6.3.9, we can determine the labels on all edges, with respect to the first or last label from the segment $S_{u,v}$, with the exception of the edge $z_j z_{j+1}$. Therefore, all edges of $S_{w,z}$ but $z_j z_{j+1}$ become determined. Since we repeat that procedure for all pairs of segments, we get that for a fixed segment $S_{w,z}$ we end up with a not yet determined edge $z_j z_{j+1}$ if and only if this is a not yet determined edge in relation to every other segment $S_{u,v}$ and its fixed vertex v_i. We repeat this procedure for all pairs of segments. Each specific calculation takes linear time. Since there are $O(k^2)$ segments, the whole calculation takes $O(k^4)$ time.

From the above procedure (where we were determining labels of edges of segments with each other) we conclude that all of the edges $e_i = v_i v_{i+1}$ of a segment $S_{u,v} = (u = v_1, v_2, \ldots, v_p = v)$ are in one of the following relations. First, where all of the edges are determined with respect to each other. Second, where there are some edges $e_1, e_2, \ldots e_{i-1}$, whose label is determined with respect to the label $\lambda(e_1)$, there is an edge $f = e_i = v_i v_{i+1}$ which is not yet determined, and then there follow the edges $e_{i+1}, e_{i+2}, \ldots, e_{p-1}$, whose labels are determined with respect to $\lambda(e_{p-1})$. Third, where the first $e_1, \ldots, e_{i-1}$ edges are determined with respect to the $\lambda(e_1)$ and all of the remaining edges $e_i, e_{i+1} \ldots, e_{p-1}$ are determined with respect to the $\lambda(e_{p-1})$. We want to now determine all of the edges in such segment $S_{u,v}$ with respect to just one edge (either the first or the last one). In the second case, we use the fact that at least one of the temporal paths between v_{i-1} and v_{i+1} has to pass through f, to determine $\lambda(f)$ with respect to $\lambda(e_{i-1})$ (and consequently $\lambda(e_1)$), and similarly, one of the temporal paths between v_i and v_{i+2} has to pass through f, which determines $\lambda(f)$ with respect to $\lambda(e_{i+1})$ (and consequently $\lambda(e_{p-1})$). In the third case, knowing the temporal paths between v_{i-1} to v_{i+1} results in determining the label of $\lambda(e_{i-1})$ with $\lambda(e_i)$, which consequently relates labels of all of the edges of the segment against each other. To determine the desired paths we proceed as follows.

G-11. Let $S_{u,v} = (u = v_1, v_2, \ldots, v_p = v)$ be a segment of length at least 4. If there is a not yet determined edge $v_i v_{i+1} = f$ in $S_{u,v}$ then we guess which of the fastest temporal paths: from v_{i-1} to v_{i+1}, from v_{i+1} to v_{i-1}, from v_i to v_{i+2}, from v_{i+2} to v_i pass through the edge f. If there are two incident edges $e = v_{i-1} v_i$ and $f = v_i v_{i+1}$ in $S_{u,v}$, that are determined with respect to $\lambda(v_1 v_2)$ and $\lambda(v_{p-1} v_p)$, respectively then we guess which of the fastest temporal paths: from v_{i-1} to v_{i+1}, from v_{i+1} to v_{i-1} pass through the edges e, f.

We create $O(1)$ guesses for every such segment $S_{u,v}$, and $O(k^2)$ new guesses in total, as there are at most $O(k^2)$ segments.

Note that the condition for segment length of at least four comes from Lemma 6.3.15. We now conclude the following.

Corollary 6.3.20. *Let $S_{u,v}$ be an arbitrary segment in G'. If $S_{u,v}$ is of length 3 or 2 then it has at most 3 or 2 not yet determined edges, respectively. If $S_{u,v}$ is of length at least 4 then the labels of all its edges are determined with respect to the first edge.*

At this point G' is a graph, where each edge e has a value for its label $\lambda(e)$ that depends on (i. e., is a function of) some other label $\lambda(f)$ of edge f, or it depends on no other label. We now describe how we create variables and start building our ILP instances. For every edge e in G' that is incident to a vertex of interest, we create a variable x_e that can have values from $\{1, 2, \ldots, \Delta\}$. Besides that, we create one variable for each edge that is still not yet determined on a segment. Since each vertex of interest is incident to at most k edges in G', and each segment has at most one extra not yet determined edge, we create $O(k^2)$ variables. At the end, we create our final guess.

G-12. We guess the permutation of all $O(k^2)$ variables. So, for any two variables x_e and x_f, we know if $x_e < x_f$ or $x_e = x_f$, or $x_e > x_f$. This results in $O(k^2!) = k^{O(k^2)}$ guesses and consequently each of the ILP instances we created up to now is further split into $k^{O(k^2)}$ new ones.

We have now finished creating all ILP instances. From Section 6.3.2 we know the structure of all guessed paths, to which we have just added also the knowledge of permutation of all variables. We proceed with adding constraints to each of our ILP instances. First, we add all constraints for the labels of edges that we have determined up to now. We then continue to iterate through all pairs of vertices and start adding equality (resp. inequality) constraints for the fastest (resp. not necessarily fastest) temporal paths between them.

We now describe how we add constraints to a path. Whenever we say that the duration of a path gives an equality or inequality constraint, we mean the following. Let $P = (u = v_1, v_2, \ldots, v_p = v)$ be the underlying path of a fastest temporal path from u to v, and let $Q = (u = z_1, z_2, \ldots, z_r = v)$ be the underlying path of another temporal path from u to v. Then we know that $d(P, \lambda) = D_{u,v}$ and $d(Q, \lambda) \geq D_{u,v}$. Using Observation 6.3.7 we create an *equality constraint* for P of the form

$$D_{u,v} = \sum_{i=2}^{p-1} (\lambda(v_i v_{i+1}) - \lambda(v_{i-1} v_i))_\Delta + 1, \tag{6.5}$$

and an *inequality constraint* for Q

$$D_{u,v} \leq \sum_{i=2}^{r-1} (\lambda(z_i z_{i+1}) - \lambda(z_{i-1} z_i))_\Delta + 1. \tag{6.6}$$

In both cases we implicitly assume that if the difference of $(\lambda(z_i z_{i+1}) - \lambda(z_{i-1} z_i))$ is not positive, for some i, we add the value Δ to it (i. e., we consider the difference modulo Δ), therefore we have the sign Δ around the brackets. Note that we can determine if the difference between two consecutive labels is positive or negative. In the case when two consecutive labels are determined with respect to the same label $\lambda(e)$ the difference between them is easy to determine. If consecutive labels are not determined with respect to the same label, both labels are considered undetermined and are assigned a variable for which we know in what kind of relation they are (see guess G-12). Therefore, we know when Δ has to be added, which implies that Equations (6.5) and (6.6) are calculated correctly for all paths.

We iterate through all pairs of vertices x, y and make sure that the fastest temporal path from x to y produces the equality constrain Equation (6.5), and all other temporal paths from x to y produce the inequality constraint Equation (6.6). For each pair, we argue how we determine these paths.

Fastest paths between $\boldsymbol{u, v \in U}$. Let $u, v \in U$, i. e., both u, v are vertices of interest. For the path from u to v (resp. from v to u) in G', which we guessed that it coincides with the fastest in G-1, we introduce an equality constraint. We then iterate over all other paths from u to v (resp. from v to u) in G', and for each one we introduce an inequality constraint. There are $k^{O(k)}$ possible paths from u to v in G'. Therefore we add $k^{O(k)}$ inequality constraints for the pair u, v.

Fastest paths from $\boldsymbol{u \in U}$ to $\boldsymbol{x \in V(G') \setminus U}$. From the guesses G-8 and G-10 we know the fastest temporal paths from u to all vertices in a segment $S_{w,v}$. In this case, we create an equality constraint for the fastest path and we iterate through all other paths, for which we introduce the inequality constraints. There are $k^{O(k)}$ possible paths of the form $u \rightsquigarrow w$ (resp. $u \rightsquigarrow v$), and a unique way how to extend these paths from w (resp. v) to reach x in $S_{w,v}$. Therefore we add $k^{O(k)}$ inequality constraints for the pair u, x.

Fastest paths from $\boldsymbol{x \in V(G') \setminus U}$ to $\boldsymbol{u \in U}$. Let x be a vertex in the segment $S_{w,z} = (w = z_1, z_2, \ldots, z_r = z)$, and let $u \in U$. If $S_{w,z}$ is of length 3 or less, then we already know the fastest temporal path from every vertex in the segment to u (since $S_{w,z}$ has at most 2 inner vertices, we determined the fastest temporal paths from them to u in guess G-4).

Assume that $S_{w,z}$ is of length at least 4. From Corollary 6.3.20 we know that the labels of all the edges in $S_{w,z}$ are determined with respect to the label of the first edge. Moreover, this gives us the knowledge of the exact differences among two consecutive edge labels, which is enough to uniquely determine travel delays at all of the inner vertices $z_i \in S_{w,z}$ (see Definition 6.3.6).

From the matrix D we can easily determine the two vertices $z_i, z_{i+1} \in S_{w,z} \setminus \{w, z\}$ for which the fastest temporal path from z_i to u has the biggest duration. Let us denote with P^+ the fastest temporal path of the form $z_2 \to z \rightsquigarrow u$, and with P^- the fastest temporal path of the form $z_{r-1} \to w \rightsquigarrow u$ (we know these paths from guess G-8). It follows that all vertices z_j in $S_{w,z} \setminus \{z_i, z_{i+1}\}$ that are closer to w than z_i, z_{i+1} reach u the fastest using the path $(z_j \to z_{j-1} \to \cdots \to z_2) \cup P^+$ and all the vertices z_j in $S_{w,z} \setminus \{z_i, z_{i+1}\}$ that are closer to z than z_i, z_{i+1} reach u the fastest using the path $(z_j \to z_{j+1} \to \cdots \to z_{r-1}) \cup P^-$. Since the first part of the above path is unique, and we know that the second part is the fastest, it follows that these paths indeed represent the fastest temporal paths to u. What remains to determine is the fastest temporal paths from z_i, z_{i+1} to u. We distinguish the following two options.

(i) $z_i \neq z_{i+1}$. Then the fastest temporal path from z_i to u is $(z_i \to z_{i-1} \to \cdots \to z_2) \cup P^+$, and the fastest temporal path from z_{i+1} to u is $(z_{i+1} \to z_{i+2} \to \cdots \to z_{r-1}) \cup P^-$.

(ii) $z_i = z_{i+1}$, i. e., let z_i be the unique vertex, that is furthest away from u in $S_{w,z}$. In this case, we have to determine if the fastest temporal path from z_i to u, travels first through vertex z_{i-1} (and then through w), or it travels first through z_{i+1} (and then through z). Since we know the values $D_{z_{i-1},u}, D_{z_{i+1},u}$, and we know the value of the waiting time $\tau_{v_{i-1}}^{v_i,v_{i-2}}$ at vertex v_{i-1} when traveling from v_i to v_{i-2}, we can uniquely determine the desired path. We set $c = D_{z_{i-1},u}+\tau_{v_{i-1}}^{v_i,v_{i-2}}$ and compare c to the value $D_{z_i,u}$. If $c < D_{z_i,u}$ we conclude that our ILP has no solution and we stop with calculations if $c = D_{z_i,u}$ then the fastest temporal path from z_i to u is of the form $(z_i \to z_{i-1} \to \cdots \to z_2) \cup P^+$, if $c > D_{z_i,u}$ then the fastest temporal path from z_i to u is of the form $(z_i \to z_{i+1} \to \cdots \to z_{r-1}) \cup P^-$.

Once the fastest temporal path from x to u is determined, we introduce an equality constraint for it. For each of the other $k^{O(k)}$ paths from x to u (which correspond to all paths of the form $w \rightsquigarrow u$ and $z \rightsquigarrow u$, together with the unique subpath on $S_{w,z}$), we introduce an inequality constraint. Therefore we add $k^{O(k)}$ inequality constraints for the pair x, u.

Fastest paths between $\boldsymbol{x, y \in V(G') \setminus U}$. Let $x, y \in V(G') \setminus U$. There are two options.

(i) Vertices x, y are in the same segment $S_{u,v} = (u, v_1, v_2, \ldots, v_p, v)$. If the length of $S_{u,v}$ is less than 4 then we know what is the fastest path between vertices, as $x, y \in U^*$. Suppose now that $S_{u,v}$ is of length at least 4 and assume that x is closer to u in $S_{u,v}$ than y. Then we have two options; either the path from x to y travels only through the edges of $S_{u,v}$, denote such path as $P_{x,y}$, or it is of the form $x \to v_1 \to u \rightsquigarrow v \to v_p \to y$, denote is as $P^*_{x,y}$. Note that we can determine $P^*_{x,y}$ as it is a concatenation of a unique path from x to v_2, together with the fastest path from v_2 to v_p, that travels through u and v (we know this path from G-7), and the unique path from v_p to y. Because of Corollary 6.3.20 we can determine $c = d(P_{x,y}, \lambda)$. If $c > D_{x,y}$ we set the fastest path to be $P^*_{x,y}$, if $c = D_{x,y}$ then the fastest path is $P_{x,y}$, and if $c < D_{x,y}$ we conclude that our ILP has no solution and we stop with calculations.

(ii) Vertices x and y are in different segments. Let x be a vertex in the segment $S_{u,v} = (u = v_1, v_2, \ldots, v_p = v)$ and let y be a vertex in the segment $S_{w,z} = (w = z_1, z_2, z_3, \ldots, z_r = z)$. By checking the durations of the fastest paths from x to every vertex in $S_{w,z} \setminus \{w, z\}$ we can determine the vertex $z_i \in S_{w,z}$, for which the duration from x is the biggest. Note that if there are two such vertices z_i and z_{i+1}, we know exactly how all fastest temporal paths enter $S_{w,z}$ (we use similar arguing as in case (i) from above, where we were determining the fastest path from $x \in V(G')$ to $u \in U$). This implies that the fastest temporal paths from x to all vertices $z_2, z_3, \ldots, z_{i-1}$ (resp. $z_{i+1}, z_{i+2}, \ldots, z_{r-1}$) pass through w (resp. z). Now we determine the vertex $v_j \in S_{u,v} \setminus \{u, v\}$, for which the value of the durations of the fastest paths from it to the vertex y is the biggest. Again, if there are two such vertices v_j and v_{j+1} we know exactly how the fastest temporal paths, starting in these two vertices, leave the segment $S_{u,v}$. We use similar arguing as in case (i) from above when we were determining the fastest path from $x \in V(G')$ to $u \in U$. Knowing the vertex v_j implies that the fastest temporal paths from the vertices $v_2, v_2, \ldots, v_{j-1}$ (resp. $v_{j+1}, v_{j+2}, \ldots, v_{p-1}$) to the vertex y passes through u (resp. v). Since we know the following fastest temporal paths (see guess G-7) $z_2 \to w \rightsquigarrow u \to v_2$, $z_2 \to w \rightsquigarrow v \to v_{p-1}$, $z_{r-1} \to z \rightsquigarrow v \to v_{p-1}$ and $z_{r-1} \to z \rightsquigarrow v \to v_{p-1}$, we can uniquely determine all fastest temporal paths from $x \neq v_j$ to any $y \in S_{u,v} \setminus \{z_i\}$.

In case when $x = v_j$ and $y = z_i$ and the segments are of length at least 4, we can uniquely determine the fastest path from v_j to z_i, using similar arguing as in case (ii) from above, when we were determining the fastest path from $x \in V(G')$ to $u \in U$. If at least one of the segments is of length 3 or less, we can again uniquely determine the fastest path from v_j to z_i, using the same approach, and the knowledge of fastest paths to (or from) all vertices of the segment of length 3 (as we guessed them in guess G-7).

Once the fastest path is determined we introduce the equality constraint for it and iterate through all other paths, for which we introduce inequality constraints. To enumerate all these non-fastest temporal paths, we just consider all possible paths $u \rightsquigarrow w$, where u and w are the vertices of interest that are the endpoints of segments to which x and y belong; once the correct segment is reached, there is a unique path to the desired vertex x (resp. y). Therefore we introduce $k^{O(k)}$ inequality constraints for each pair of vertices x, y.

Fastest paths for vertices in Z. All of the above is enough to determine the labeling λ of G'. We have to extend the labeling to consider also the vertices of $Z = V(G) \setminus V(G')$ that we initially removed from G.

Recall that $G[Z]$ consists of disjoint trees and that each of these trees has a unique neighbor (clip vertex) v in G'. We then define the tree T_v in $G[Z \cup \{v\}]$ as the collection of trees from $G[Z]$ with a clip vertex v, together with the root v. Determining the fastest temporal paths between any two vertices in the same tree is a straightforward process (see Theorem 6.3.1), therefore we exclude this case from our upcoming analysis. From Observation 6.3.19 it follows that knowing temporal paths between any $y \in V(G')$ and all vertices in the first layer of the tree T_v (i. e., children of the root v), it is enough to determine the fastest temporal paths between y and all other vertices in T_v. Therefore, in the upcoming analysis, we focus only on the vertices in the first layer of each tree T_v. During the process of determining the fastest paths from and to the vertices in Z, we use the fact that we have already identified the fastest paths among all vertices in G'.

We split our analysis into two cases. First, when the clip vertex v of tree T_v is not a vertex of interest, and second when the clip vertex is also a vertex of interest in G'. In the first case, we use the fact that v has only two edges e, f incident to it in G', and that we can determine all the labels of the tree edges with respect to $\lambda(e), \lambda(f)$ (see Lemma 6.3.17). This turns out to be enough for us to determine the fastest temporal paths among any vertex r in the first layer of the tree T_v and an arbitrary vertex in $V(G) \setminus V(T_v)$. In the second case, we cannot determine the labeling of the tree with respect to the labels of all edges incident to the clip vertex. Therefore, we split the vertices in the first layer of T_v into equivalence classes, and use the fact that the fastest temporal paths between two vertices in the same equivalence class coincide on the edges outside of T_v.

Fastest paths from $\boldsymbol{r \in Z}$ to $\boldsymbol{y \in U \cup U^*}$. Let $x \in V(G')$ be a clip vertex of the tree T_x in $G[Z \cup \{x\}]$ with $r \in V(T_x)$ being a vertex in the first layer of T_x. We distinguish the following two cases.

(i) The clip vertex $x = u \in U$ is a vertex of interest. In this case, we can w.l.o.g. assume that r is a representative vertex in its equivalence class among the first layer vertices of T_u. From the guesses G-5 and G-6 we know the fastest temporal path from r to y.

(ii) The clip vertex $x \in U$ is not a vertex of interest. Then $x = v_j$ is a part of some segment $S_{u,v} = \{u = v_1, v_2, \dots, v_p = v)$, where $j \neq 1 \neq p$. Using Lemma 6.3.17 we can determine all the edge labels of T_x with respect to the label $\lambda(v_{j-1}v_j)$ and with respect to the label $\lambda(v_j v_{j+1})$. Using the calculations of fastest temporal paths among vertices in G' and the performed guesses we know the exact structure (i. e., the sequence of vertices and edges) of the following paths:

- path P^*_{xy} which is the fastest temporal path from the vertex x to the vertex y,
- path P^u_{xy} which is the fastest temporal path from the vertex x to the vertex y, that passes through the vertex u,
- path P^v_{xy} which is the fastest temporal path from the vertex x to the vertex y, that passes through the vertex v.

Note that P^*_{xy} is either equal to the path P^u_{xy} or to the path P^v_{xy}. More precisely, from the guesses performed we know the structure of the fastest path from v_2 through u, which then continues to any other vertex of interest, and any other neighbor of the vertex of interest (see guesses G-7 and G-8). This path can then be easily (uniquely) extended to start from $x = v_i$, as there is a unique (temporal) path starting at x and finishing at u or v.

Suppose now that $P^*_{xy} = P^u_{xy}$. Since the labels of T_x are determined with respect to $\lambda(v_{i-1}x)$ we can calculate the value $c = D_{x,y} + |\lambda(v_{i-1}x) - \lambda(rx)|$. We then compare c to $D_{r,y}$ and get one of the following three options. First $c = D_{r,y}$, in this case, the fastest temporal path from r to y uses first the edge rx and then continues to y using the edges and vertices of P^*_{xy}. Second $c > D_{r,y}$, in this case, the fastest temporal path from r to y uses first the edge rx and then continues to y using the vertices and edges of P^v_{xy}. Third $c < D_{r,y}$, in this case, we stop the calculation and return false, as it cannot happen that a temporal path has a smaller duration than the corresponding value in the matrix D.

In both cases, we introduce an equality constraint for the determined fastest temporal path and inequality constraints for all the other $k^{O(k)}$ paths.

Fastest paths from $\boldsymbol{r \in Z}$ to $\boldsymbol{y \in V(G) \setminus (U \cup U^*)}$. The proof in this case is similar to the one above. We still split the analysis into two parts, one where the clip vertex x of a tree T_x that includes r is in U and one where it is not in U. The difference is that in some cases we need to also extend the ending part of the path (which can be done uniquely, using the same arguments as in the above analysis).

Once we determine the fastest temporal path from r to y we introduce an equality constraint for it and for all other $k^{O(k)}$ paths we introduce inequality constraints.

The procedure produces one equality constraint (for the fastest path) and $k^{O(k)}$ inequality constraints.

Fastest paths from $\boldsymbol{y \in V(G)}$ to $\boldsymbol{r \in Z}$. The process of determining fastest temporal paths from any vertex in the graph G to a vertex r that is a vertex in the first layer of a tree $T_x \in G[Z \cup \{x\}]$, where $x \in V(G')$, is similar to the one above, but performed in the opposite direction.

Solving ILP instances

All of the above finishes our construction of ILP instances. We have created $f(k)$ instances (where f is a double exponential function), each with $O(k^2)$ variables and $O(n^2)g(k)$ constraints (again, g is a double exponential function). We now solve each ILP instance I, using results from Lenstra [92], in the FPT time, with respect to k. If none of the ILP instances gives a positive solution, then there exists no labeling λ of G that would realize the matrix D (i. e., for any pair of vertices $u, v \in V(G)$ the duration of a fastest temporal path from u to v has to be $D_{u,v}$). If there is at least one I that has a valid solution, we use this solution and produce our labeling λ, for which (G, λ) realizes the matrix D. We have proven in the previous subsections that this is true since each ILP instance corresponds to a specific configuration of fastest temporal paths in the graph (i. e., considering all ILP instances is equivalent to exhaustively searching through all possible temporal paths between vertices). Besides that, in each ILP instance, we add also the constraints for durations of all temporal paths between each pair of vertices. This results in setting the duration of a fastest path from a vertex $u \in V(G)$ to a vertex $v \in V(G)$ as $D_{u,v}$, and the duration of all other temporal paths from u to v, to be greater or equal to $D_{u,v}$, for all pairs of vertices u, v. Therefore, if there is an instance with a positive solution, then this instance gives rise to the desired labeling, as it satisfies all of the constraints. For the other direction, we can observe that if there is a labeling λ meeting all duration requirements specified by D, then this labeling produces a specific configuration of fastest temporal paths. Since we consider all configurations, one of the produced ILP instances will correspond to the configuration implicitly defined by λ, and hence our algorithm finds a solution.

To create the labeling λ from a solution X, of a positive ILP instance, we use the following procedure. First we label each edge e, that corresponds to the variable x_e by assigning the value $\lambda(e) = x_e$. We then continue to set the labels of all other edges. We know that the labels of all of the remaining edges depend on the label of (at least one) of the edges that were determined in the previous step. Therefore, we easily calculate the desired labels for all remaining edges.

6.4 Concluding remarks

We have introduced a natural and canonical temporal version of the graph realization problem with respect to distance requirements, called Simple periodic Temporal Graph Realization. We have shown that the problem is NP-hard in general and polynomial-time solvable if the underlying graph is a tree. Building upon those results, we have investigated its parameterized computational complexity with respect to structural parameters of the underlying graph that measure "tree-likeness". For those parameters, we essentially gave a tight classification between parameters that allow for tractability (in the FPT sense) and parameters that presumably do not. We showed that our problem is W[1]-hard when parameterized by the feedback vertex number of the underlying graph, and that it is in FPT when parameterized by the feedback edge number of the underlying graph. Note that most other common parameters that measure tree-likeness (such as the treewidth) are smaller than the feedback vertex number.

We believe that our work spawns several interesting future research directions and builds a base upon which further temporal graph realization problems can be investigated.

Further parameterizations. There are several structural parameters which can be considered to obtain tractability which are either larger or incomparable to the feedback vertex number.

- The *vertex cover number* measures the distance to an independent set, on which we trivially only have no-instances of our problem. We believe this is a promising parameter to obtain tractability.
- The *tree-depth* measures "star-likeness" of a graph and is incomparable to both the feedback vertex number and the feedback edge number. We leave the parameterized complexity of our problem with respect to this parameter open.
- Parameters that measure "path-likeness" such as the *pathwidth* or the *vertex deletion distance to disjoint paths* are also natural candidates to investigate.

Furthermore, we can consider combining a structural parameter with Δ. Our NP-hardness reduction (Theorem 6.2.1) produces instances with constant Δ, so as a single parameter Δ cannot yield fixed-parameter tractability. However, in our parameterized hardness reduction (Theorem 6.2.2) the value for Δ in the produced instance is large. This implies that our result does not rule out e.g. fixed-parameter tractability for the combination of the treewidth and Δ as a parameter. We believe that investigating such parameter combinations is a promising future research direction.

Further problem variants. There are many natural variants of our problem that are well-motivated and warrant consideration. In the following, we give two specific examples. We believe that one of the most natural generalizations of our problem is to allow more than one label per edge in every Δ-period. A well-motivated variant (especially from the network design perspective) of our problem would be to consider the entries of the duration matrix D as upper-bounds on the duration of fastest paths rather than exact durations. Our work gives a starting point for many interesting future research directions such as the two mentioned examples.

CHAPTER 7

Conclusion

In this dissertation, we explored the complex domain of temporal graphs, with the central goal of contributing to the development and understanding of the fast-growing field of temporal graphs. We achieved this by presenting a detailed study of four different problems.

We started by addressing the problem of finding temporally disjoint paths in a temporal graph with an underlying graph forming a path or a tree. We proved that the problem is NP-hard in general, but admits an FPT algorithm when parameterized by the desired number of disjoint paths. Restricting the problem even further, focusing on the case where the disjoint paths must traverse all the vertices of the underlying path graph, we provided a polynomial-time algorithm for this specific scenario. Our work left interesting research questions unanswered, which were further explored and addressed by Kunz et al. [91], providing valuable insights into the parameterized computational complexity of the problem.

The next problems we studied were problems Temporal Vertex Cover (TVC) and Sliding-Window Temporal Vertex Cover (Δ-TVC), which represent natural extensions of the classic Vertex Cover problem. Our work was built on the initial results of Akrida et al. [5]. We showed that Δ-TVC is NP-hard already when the underlying graph of the input temporal graph is a path or cycle, and provided a *Polynomial-Time Approximation Scheme* (PTAS) for it. This provided a sharp contrast to the more tractable TVC in similar scenarios. We presented also an exact algorithm for Δ-TVC with exponential running time dependency on the number of edges of the underlying graph. This algorithm was then used as a subroutine in our polynomial-time $(d-1)$-approximation algorithm, where d is the maximum vertex degree at any time-step of the input temporal graph. This result improves the d-approximation algorithm proposed by Akrida et al. [5], and answered an open question the authors posed. We finished our study by presenting a fixed-parameter tractable algorithm, with respect to the size of an optimum solution.

The third set of problems we explored were temporal design problems for ensuring connectivity in undirected temporally connected graphs. The core objective of these optimization problems was: given an undirected graph G, what is the smallest number $|\lambda|$ of time-labels needed to assign to the edges of G such that (G, λ) is temporally connected (i. e., there is a temporal path among each pair of vertices)? We presented scenarios where this task is computationally challenging and other ones where it demonstrates a more tractable behaviour, which led us to investigate the complex connection between time and structure in these graphs. We showed that the unrestricted problem, called MIN. LABELING (ML) can be solved in polynomial time. We then continued with the NP-hardness of MAL when the required maximum age is equal to the diameter d_G of the input static graph G. We prove that MSL is NP-hard and provide a fixed-parameter tractable algorithm for it, with respect to the size of the labeling and number of terminals.

Our final problem explored temporal graph realization and serves as an initiation of this study in the field of temporal graphs. In our work, we attempted to extend the idea of the graph realization problem with respect to vertex distances in the context of temporal graphs. We focused our study on periodic temporal graphs, i. e., temporal graphs in which the temporal availability of each edge of the underlying graph is periodic. In the SIMPLE TGR problem, we were given a matrix of the fastest path durations between all vertices and were tasked with determining the labeling of a periodic temporal graph that would realize these durations. We showed that while this problem is generally hard (NP-hard already for a small constant Δ), it became more manageable when the underlying structure resembled a tree (polynomial-time solvable if the underlying graph G is a tree and fixed-parameter tractable, with respect to the feedback edge number of G).

Altogether, this dissertation contributes to the development of techniques for analyzing and solving problems involving temporal graphs. Through this work, we offer some valuable perspectives for managing the complexities of graphs evolving over time, encouraging further exploration in this dynamic field.

Bibliography

[1] Eric Aaron, Danny Krizanc, and Elliot Meyerson. DMVP: foremost waypoint coverage of time-varying graphs. In *Proceedings of the 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 29–41, 2014.

[2] A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[3] Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. Ephemeral networks with random availability of links: The case of fast networks. *Journal of Parallel and Distributed Computing*, 87:109–120, 2016.

[4] Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61(3):907–944, 2017.

[5] Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107:108–123, 2020.

[6] Paola Alimonti and Viggo Kann. Hardness of approximating problems on cubic graphs. In *Proceedings of the 3rd Italian Conference on Algorithms and Complexity (CIAC)*, pages 288–298, 1997.

[7] Shaull Almagor and Morteza Lahijanian. Explainable multi agent path finding. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20*, pages 34–42, 2020.

[8] Emmanuel Arrighi, Niels Grüttemeier, Nils Morawietz, Frank Sommer, and Petra Wolf. Multi-parameter analysis of finding minors and subgraphs in edge-periodic temporal graphs. In *Proceedings of the 48th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 283–297, 2023.

[9] Dor Atzmon, Roni Stern, Ariel Felner, Glenn Wagner, Roman Barták, and Neng-Fa Zhou. Robust multi-agent path finding and executing. *Journal of Artificial Intelligence Research*, 67:549–579, 2020.

[10] John Augustine, Keerti Choudhary, Avi Cohen, David Peleg, Sumathi Sivasubramaniam, and Suman Sourav. Distributed graph realizations. *IEEE Transactions on Parallel and Distributed Systems*, 33(6):1321–1337, 2022.

[11] Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Efficiently realizing interval sequences. *SIAM Journal on Discrete Mathematics*, 34(4):2318–2337, 2020.

[12] Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Graph realizations: Maximum degree in vertex neighborhoods. In *Proceedings of the 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 10:1–10:17, 2020.

[13] Amotz Bar-Noy, David Peleg, Mor Perry, and Dror Rawitz. Composed degree-distance realizations of graphs. In *Proceedings of the 32nd International Workshop on Combinatorial Algorithms (IWOCA)*, pages 63–77, 2021.

[14] Amotz Bar-Noy, David Peleg, Mor Perry, and Dror Rawitz. Graph realization of distance sets. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 13:1–13:14, 2022.

[15] Mehdi Behzad and James E Simpson. Eccentric sequences and eccentric sets in graphs. *Discrete Mathematics*, 16(3):187–193, 1976.

[16] Matthias Bentert, Anne-Sophie Himmel, Hendrik Molter, Marco Morik, Rolf Niedermeier, and René Saitenmacher. Listing all maximal k-plexes in temporal graphs. *ACM Journal of Experimental Algorithmics*, 24(1):13:1–13:27, 2019.

[17] Matthias Bentert, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Applied Network Science*, 5(1):73, 2020.

[18] Kenneth A. Berman. Vulnerability of scheduled networks and a generalization of Menger's Theorem. *Networks*, 28(3):125–134, 1996.

[19] René van Bevern, Matthias Mnich, Rolf Niedermeier, and Matthias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, 2015.

[20] Robert E Bixby and Donald K Wagner. An almost linear-time algorithm for graph realization. *Mathematics of Operations Research*, 13(1):99–123, 1988.

[21] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[22] Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

[23] Richard T. Bumby. A problem with telephones. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):13–18, 1981.

[24] Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Formal Models and Metrics. Technical report, Defence R&D Canada, April 2013. URL: https://hal.archives-ouvertes.fr/hal-00865762.

[25] Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Problems, Analysis, and Algorithmic Tools. Technical report, Defence R&D Canada, April 2013. URL: https://hal.archives-ouvertes.fr/hal-00865764.

[26] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[27] Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021.

[28] Jiehua Chen, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. Cluster editing in multi-layer and temporal graphs. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC)*, volume 123, pages 24:1–24:13, 2018.

[29] Wai-Kai Chen. On the realization of a (p, s)-digraph with prescribed degrees. *Journal of the Franklin Institute*, 281(5):406–422, 1966.

[30] Fan Chung, Mark Garrett, Ronald Graham, and David Shallcross. Distance realization problems with applications to internet tomography. *Journal of Computer and System Sciences*, 63(3):432–448, 2001.

[31] Julia Chuzhoy, David Hong Kyun Kim, and Rachit Nimavat. New hardness results for routing on disjoint paths. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 86–99, 2017.

[32] Andrea E. F. Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. Flooding time of edge-markovian evolving graphs. *SIAM Journal on Discrete Mathematics*, 24(4):1694–1712, 2010.

[33] Joseph C. Culberson and Piotr Rudnicki. A fast algorithm for constructing trees from distance matrices. *Information Processing Letters*, 30(4):215–220, 1989.

[34] Marek Cygan, Fedor V. Fomin, Ł ukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, MichałPilipczuk, and Saket Saurabh. *Parameterized algorithms.* Springer, Cham, 2015.

[35] Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. In *Computing and combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 216–225. Springer, Berlin, 2010.

[36] Argyrios Deligkas, Eduard Eiben, and George Skretas. Minimizing reachability times on temporal graphs via shifting labels. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 5333–5340, 2023.

[37] Riccardo Dondi and Florian Sikora. Finding disjoint paths on edge-colored graphs: more tractability results. *Journal of Compinatorial Optimization*, 36:1315–1332, 2017.

[38] R. G. Downey and M. R. Fellows. *Parameterized complexity.* Monographs in Computer Science. Springer-Verlag, New York, 1999.

[39] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, London, 2013.

[40] S.E. Dreyfus and R.A. Wagner. The steiner problem in graphs. *Networks*, 1:195–207, 1971.

[41] Jessica Enright and Kitty Meeks. Deleting edges to restrict the size of an epidemic: a new application for treewidth. *Algorithmica*, 80(6):1857–1889, 2018.

[42] Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021.

[43] Jessica Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *Journal of Computer and System Sciences*, 115:169–186, 2021.

[44] Jessica A. Enright, Kitty Meeks, and Hendrik Molter. Counting temporal paths. In *Proceedings of the 40th International Symposium on Theoretical Aspects of Computer Science (STACS '23)*, volume 254 of *LIPIcs*, pages 30:1–30:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[45] Paul Erdős and Tibor Gallai. Graphs with prescribed degrees of vertices. *Mat. Lapok*, 11:264–274, 1960.

[46] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *Journal of Computer and System Sciences*, 119:1–18, 2021.

[47] Thomas Erlebach and Jakob T. Spooner. A game of cops and robbers on graphs with periodic edge-connectivity. In *Proceedings of the 46th International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, pages 64–75, 2020.

[48] Michael R. Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

[49] Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.

[50] A. Ferreira. Building a reference combinatorial model for manets. *IEEE Network*, 18(5):24–29, 2004.

[51] Paola Flocchini, Bernard Mans, and Nicola Santoro. Exploration of periodically varying graphs. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 534–543, 2009.

[52] Till Fluschnik, Stefan Kratsch, Rolf Niedermeier, and Manuel Sorge. The parameterized complexity of the minimum shared edges problem. *Journal of Computer and System Sciences*, 106:23–48, 2019.

[53] Till Fluschnik, Marco Morik, and Manuel Sorge. The complexity of routing with collision avoidance. *Journal of Computer and System Sciences*, 102:69–86, 2019.

[54] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10(2):111–121, 1980.

[55] András Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM Journal on Discrete Mathematics*, 5(1):25–53, 1992.

[56] András Frank. Connectivity augmentation problems in network design. *Mathematical Programming: State of the Art 1994*, 1994.

[57] H. Frank and Wushow Chou. Connectivity considerations in the design of survivable networks. *IEEE Transactions on Circuit Theory*, 17(4):486–490, 1970.

[58] Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Delay-robust routes in temporal graphs. In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 30:1–30:15, 2022.

[59] D.R. Fulkerson. Zero-one matrices with zero trace. *Pacific Journal of Mathematics*, 10(3):831–836, 1960.

[60] Subhankar Ghosal and Sasthi C Ghosh. Channel assignment in mobile networks based on geometric prediction and random coloring. In *Proceedings of the 40th IEEE Conference on Local Computer Networks (LCN)*, pages 237–240, 2015.

[61] George Giakkoupis, Thomas Sauerwald, and Alexandre Stauffer. Randomized rumor spreading in dynamic graphs. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 495–507, 2014.

[62] P.A. Golovach and D.M. Thilikos. Paths of bounded length and their cuts: parameterized complexity and algorithms. *Discrete Optim.*, 8(1):72–86, 2011.

[63] Peter A. Golovach and George B. Mertzios. Graph editing to a given degree sequence. *Theoretical Computer Science*, 665:1–12, 2017.

[64] Martin Charles Golumbic and Ann N. Trenk. *Tolerance Graphs.* Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2004.

[65] Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[66] Martin Grötschel, Clyde L Monma, and Mechthild Stoer. Design of survivable networks. *Handbooks in Operations Research and Management Science*, 7:617–672, 1995.

[67] Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 162–173, 2004.

[68] L. Guo, Y. Deng, K. Liao, Q. He, T. Sellis, and Z. Hu. A fast algorithm for optimally finding partially disjoint shortest paths. In *IJCAI*, pages 1456–1462, 2018.

[69] F. Göbel, J.Orestes Cerdeira, and H.J. Veldman. Label-connected graphs and the gossip problem. *Discrete Mathematics*, 87(1):29–40, 1991.

[70] S. Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. I. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):496–506, 1962.

[71] S. Louis Hakimi and Stephen S. Yau. Distance matrix of a graph and its realizability. *Quarterly of applied mathematics*, 22(4):305–317, 1965.

[72] Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30, 1935.

[73] Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs. In *Proceedings of the 36th Conference on Artificial Intelligence (AAAI)*, pages 10193–10201, 2022.

[74] Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs, 2022. arXiv: 2204.04832.

[75] Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[76] Klaus Heeger, Danny Hermelin, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Dvir Shabtay. Equitable scheduling on a single machine. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 11818–11825. AAAI Press, 2021.

[77] Pavol Hell and David Kirkpatrick. Linear-time certifying algorithms for near-graphical sequences. *Discrete Mathematics*, 309(18):5703–5713, 2009.

[78] Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35:1–35:16, 2017.

[79] Can Umut Ileri, Cemil Aybars Ural, Orhan Dagdeviren, and Vedat Kavalci. On vertex cover problems in distributed systems. In *Advanced Methods for Complex Network Analysis*, pages 1–29. IGI Global, 2016.

[80] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[81] Richard M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

[82] Vedat Kavalci, Aybars Ural, and Orhan Dagdeviren. Distributed vertex cover algorithms for wireless sensor networks. *International Journal of Computer Networks & Communications*, 6(1):95–110, 2014.

[83] K. Kawarabayashi, Y. Kobayashi, and B. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory, Ser. B*, 102(2):424 – 435, 2012.

[84] David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

[85] Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: Temporally disjoint paths. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4090–4096, 2021.

[86] Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: Temporally disjoint paths. *Autonomous Agents and Multi-Agent Systems*, 37(1):1, 2023.

[87] Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 62:1–62:15, 2022.

[88] Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity, 2022. arXiv:2202.0588.

[89] Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Realizing temporal graphs from fastest travel times, 2023. arXiv:2302.08860.

[90] Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Temporal graph realization from fastest paths. In *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, pages 14:1–14:16, 2024.

[91] Pascal Kunz, Hendrik Molter, and Meirav Zehavi. In which graph structures can we efficiently find temporally disjoint paths and walks? In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 180–188, 2023.

[92] Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[93] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.

[94] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[95] Linda Lesniak. Eccentric sequences in graphs. *Periodica Mathematica Hungarica*, 6:287–293, 1975.

[96] Ross M. McConnell and Jeremy P. Spinrad. Construction of probe interval models. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 866–875, 2002.

[97] F.R. McMorris, Chi Wang, and Peisen Zhang. On probe interval graphs. *Discrete Applied Mathematics*, 88(1):315–324, 1998. Computational Molecular Biology DAM - CMB Series.

[98] George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019.

[99] George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing maximum matchings in temporal graphs. *J. Comput. Syst. Sci.*, 137:1–19, 2023.

[100] George B. Mertzios, Hendrik Molter, and Viktor Zamaraev. Sliding window temporal graph coloring. *Journal of Computer and System Sciences*, 120:97–115, 2021.

[101] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016.

[102] Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016.

[103] Nils Morawietz, Carolin Rehs, and Mathias Weller. A timecop's work is harder than you think. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2020.

[104] Nils Morawietz and Petra Wolf. A timecop's chase around the table. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2021.

[105] Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry. An International Journal of Mathematics and Computer Science*, 44(4):883–895, 2010.

[106] Robertson Neil and Paul D. Seymour. Disjoint paths—a survey. *Society for Industrial and Applied Mathematics. Journal on Algebraic and Discrete Methods*, 6(2):300–305, 1985.

[107] Robertson Neil and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

[108] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2006.

[109] James B. Orlin. The complexity of dynamic languages and dynamic optimization problems. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, page 218–227, 1981.

[110] James B. Orlin. Some problems on dynamic/periodic graphs. In *Progress in combinatorial optimization (Waterloo, Ont., 1982)*, pages 273–293. Academic Press, Toronto, ON, 1984.

[111] A.N. Patrinos and S. Louis Hakimi. The distance matrix of a graph and its tree realization. *Quarterly of Applied Mathematics*, 30:255–269, 1972.

[112] Silvia Richter, Malte Helmert, and Charles Gretton. A stochastic local search approach to vertex cover. In *Proceedings of the 30th Annual German Conference on Artificial Intelligence (KI)*, pages 412–426, 2007.

[113] Elena Rubei. Weighted graphs with distances in given ranges. *Journal of Classification*, 33:282—-297, 2016.

[114] Rafael F. Santos, Alessandro Andrioni, Andre C. Drummond, and Eduardo C. Xavier. Multicolour paths in graphs: NP-hardness, algorithms, and applications on routing in WDM networks. *Journal of Combinatorial Optimization*, 33(2):742–778, 2017.

[115] Piotr Sapiezynski, Arkadiusz Stopczynski, Radu Gatej, and Sune Lehmann. Tracking human mobility using wifi signals. *PloS one*, 10(7):e0130824, 2015.

[116] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.

[117] A. Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM J. Discrete Math.*, 24(1):146–157, 2010.

[118] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*, pages 173–178, 2010.

[119] Roni Stern. Multi-agent path finding - an overview. In *Artificial Intelligence - 5th RAAI Summer School*, pages 96–115, Dolgoprudny, Russia, 2019. Springer.

[120] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Barták. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019*, pages 151–159, 2019.

[121] H. Tamura, M. Sengoku, S. Shinoda, and T. Abe. Realization of a network from the upper and lower bounds of the distances (or capacities) between vertices. In *Proceedings of the 1993 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2545—-2548, 1993.

[122] John Kit Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. Characterising temporal distance and reachability in mobile and online social networks. *Computer Communication Review*, 40(1):118–124, 2010.

[123] Binglin Tao, Mingyu Xiao, and Jingyang Zhao. Finding minimum-weight link-disjoint paths with a few common nodes. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 938–945, 2020.

[124] Suhas Thejaswi, Juho Lauri, and Aristides Gionis. Restless reachability problems in temporal graphs. *arXiv preprint 2010.08423*, 2020.

[125] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag, Berlin, 2001.

[126] Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.

[127] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[128] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[129] Bang Ye Wu. On the maximum disjoint paths problem on edge-colored graphs. *Discrete Optimization*, 9(1):50–57, 2012.

[130] Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2927–2942, 2016.

[131] Feng Yu, Amotz Bar-Noy, Prithwish Basu, and Ram Ramanathan. Algorithms for channel assignment in mobile wireless networks using temporal coloring. In *Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems (MSWiM)*, pages 49–58, 2013.

[132] Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020.